

Eusko Jaurlaritzaren Informatika Elkartea
Sociedad Informática del Gobierno Vasco

# PMD

## Manual de usuario

**Fecha: 07/08/2007**

**Referencia:**


EJIE S.A.

Mediterráneo, 3

Tel. 945 01 73 00*

Fax. 945 01 73 01

01010 Vitoria-Gasteiz

Posta-kutxatila / Apartado: 809

01080 Vitoria-Gasteiz

www.ejie.es

# Control de documentación

Título de documento: PMD

## Histórico de versiones

Código:

Versión: 1.1

Fecha: 30/04/2009

Resumen de cambios: Uso de NOPMD. Actualización a versión 4.0

Versión: 1.0

Fecha: 07/08/2007

Resumen de cambios: Primera versión.

## Cambios producidos desde la última versión

Primera versión.

## Control de difusión

Responsable: Ander Martínez

Aprobado por: Ander Martínez

Firma: Fecha:

Distribución:

## Referencias de archivo

Autor: Consultoría de áreas de conocimiento

Nombre archivo: PMD. Manual de usuario vn.n.doc

Localización:




# Contenido


| Capítulo/sección | Página |
|---|---|
| 1 Introducción | 5 |
| 2 Conceptos básicos | 5 |
| 3 Integración con Eclipse | 5 |
| 3.1 Configuración básica | 5 |
| 3.1.1. Gestión de las reglas | 7 |
| 3.1.2. Exportar reglas | 9 |
| 3.1.3. Importar reglas | 10 |
| 3.2 Parametrización de PMD para el proyecto | 11 |
| 3.2.1. Definir “Working set” | 13 |
| 3.3 Ejecución de PMD | 17 |
| 3.3.1. Generar informes | 18 |
| 3.3.2. Eliminar incidencias revisadas (Uso de NOPMD) | 20 |
| 3.3.3. Buscar código redundante (CPD) | 21 |
| 3.3.4. Eliminar incidencias | 22 |
| 3.3.5. Chequear código | 22 |
| 4 Utilidad práctica | 24 |
| 4.1 Aportaciones de PMD | 24 |
| 4.2 Aportaciones personalizadas | 25 |
| 4.3 Creación de reglas de validación personalizadas | 25 |
| 4.3.1. Implementación de una regla personalizada a través de una clase Java | 29 |
| 4.3.2. Implementación de una regla personalizada con una expresión XPath | 31 |
| 4.3.3. Cómo usar un conjunto de reglas personalizadas con Eclipse | 32 |
| 5 Tareas ant en servidor | 37 |
| 6 Anexo 1: Ejemplo | 38 |
| 6.1 Resolución | 38 |

# 1 Introducción

En este manual se describen los distintos aspectos que debe conocer el usuario sobre PMD y sobre su integración con Eclipse, así como los pasos a seguir para la creación y uso de reglas personalizadas.

# 2 Conceptos básicos

PMD es una herramienta de auditoría y verificación de código desarrollada por SourceForge.

PMD escanea el código Java y busca posibles problemas en potencia como pueden ser:

- Posibles bugs (sentencias *try/catch/finally/switch* vacias)
- Código muerto (variable locales, parámetros y métodos privados que no se usan)
- Código no óptimo (derroches en el uso de String/StringBuffer)
- Expresiones excesivamente complicadas (sentencias *if* innecesarias, implementaciones con bucles *while*)
- Código duplicado (código copiar-pegar que significa errores copiados-pegados)

PMD puede instalarse en servidor para ser ejecutado desde tareas ant, o como un plug-in de Eclipse en PC local.

Para obtener información adicional sobre el producto acceder a su página Web:

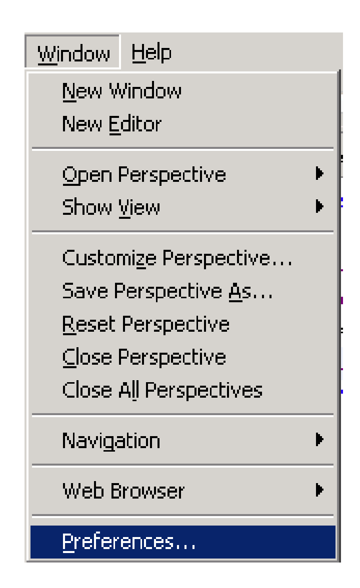
http://pmd.sourceforge.net/

# 3 Integración con Eclipse

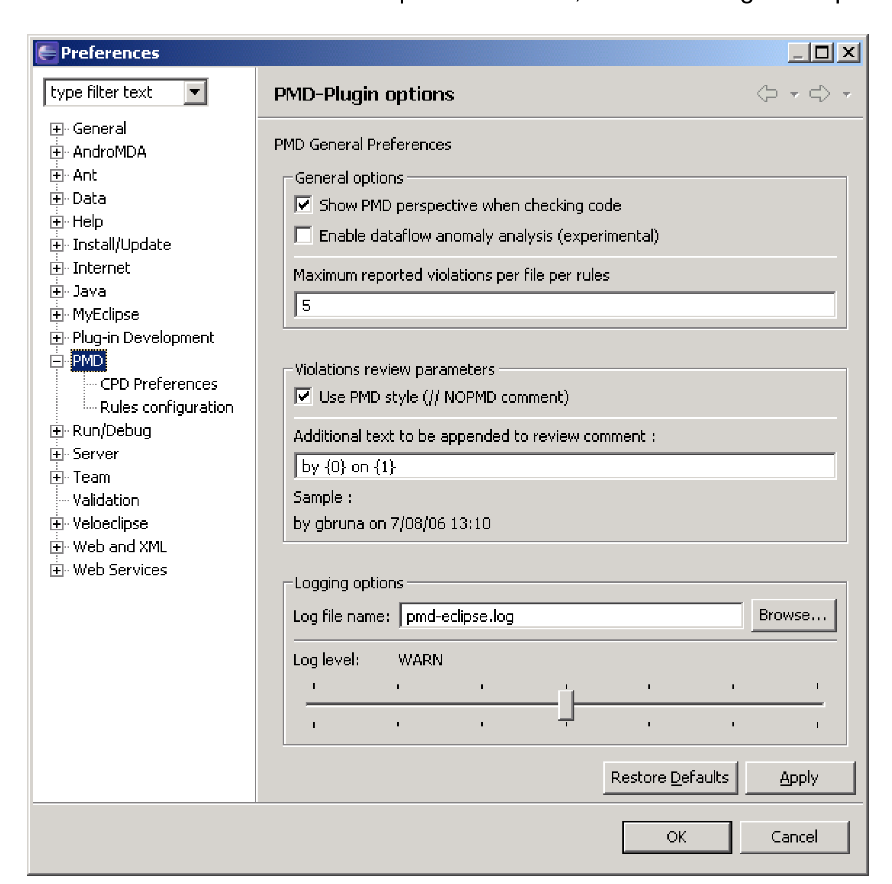
Al igual que en otros muchos casos, la integración del PMD (como plug-in) con Eclipse (o cualquier otro IDE de desarrollo) da como resultado un entorno de trabajo visual, sencillo de usar e intuitivo. Gracias a este entorno, el manejo de los aspectos de configuración y las distintas funcionalidades del PMD se realiza de forma rápida y eficaz.

## 3.1 Configuración básica

Para poder configurar el plug-in de PMD bajo Eclipse, accedemos al menú “Window→ Preferences…”




Una vez hemos accedido a la opción de menú, muestra la siguiente pantalla:




Se accede al apartado indicado como PMD, pudiendo configurar entonces preferencias generales como los parámetros de revisión de violaciones, y las opciones del fichero de Log.

PMD permite crear nuevas reglas, quitar reglas, editar reglas existentes, importar y exportar reglas, y

limpiar las existentes.

Antes de proceder a realizar cualquier tarea de importación o de edición de reglas, es aconsejable realizar una copia de seguridad de las existentes, para evitar posibles errores o confusiones y así poder deshacer cualquier cambio efectuado.

### 3.1.1. Gestión de las reglas

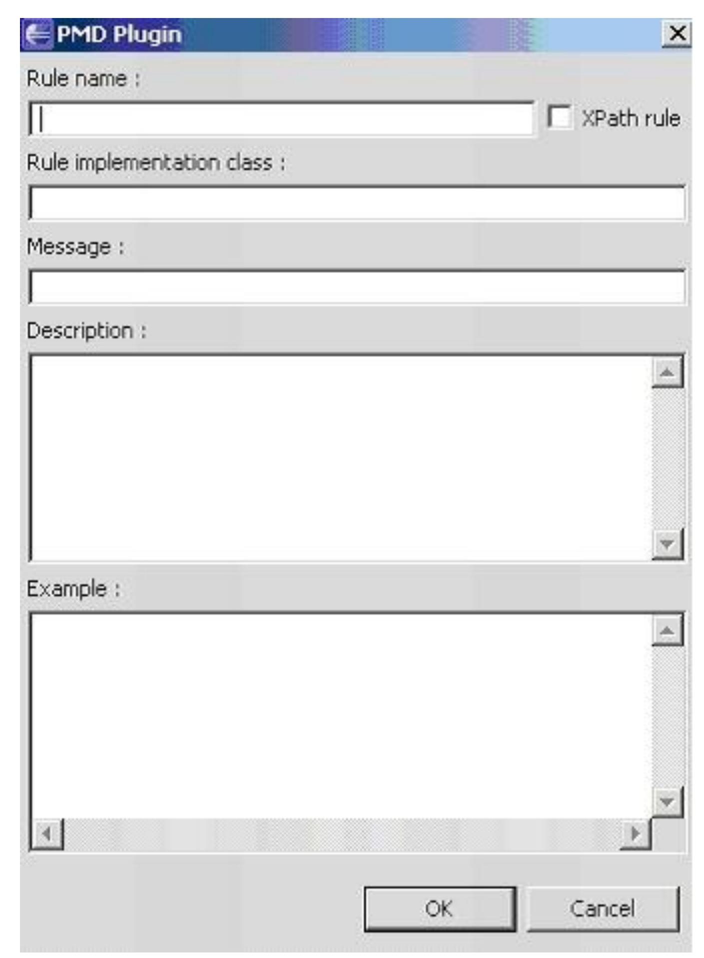
**Añadir una nueva regla**

Desde la ventana de preferencias (Window→ Preferences...), pulsamos la línea "PMD→ Rules configuration" que permite gestionar las reglas a aplicar a nuestros proyectos.

Existen dos maneras de crear una nueva regla:

- Usando Java.
- Usando una expresión XPath.

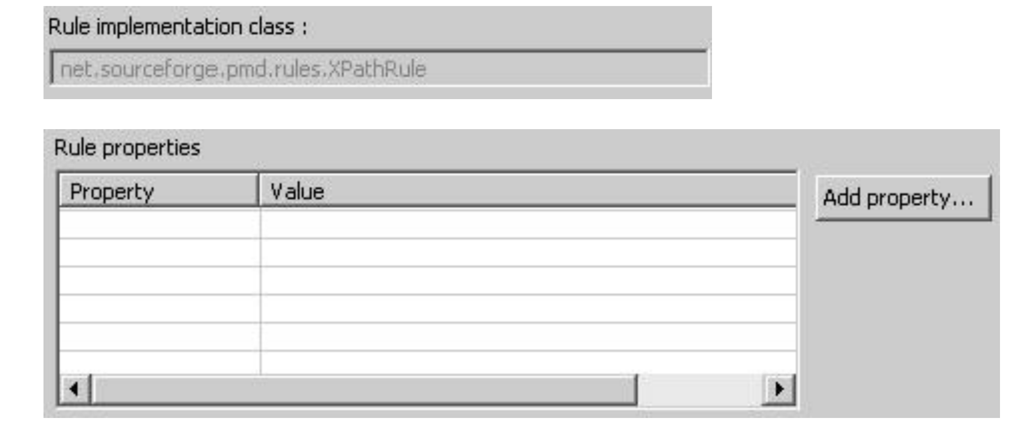
1. Pulsar el botón "Add rule…"




2. Introducir el nombre de la regla, el mensaje, la descripción y el ejemplo. Además:

a. En caso de haber creado la regla mediante una clase Java indicar cuál es en el campo *Rule Implementation Class*.

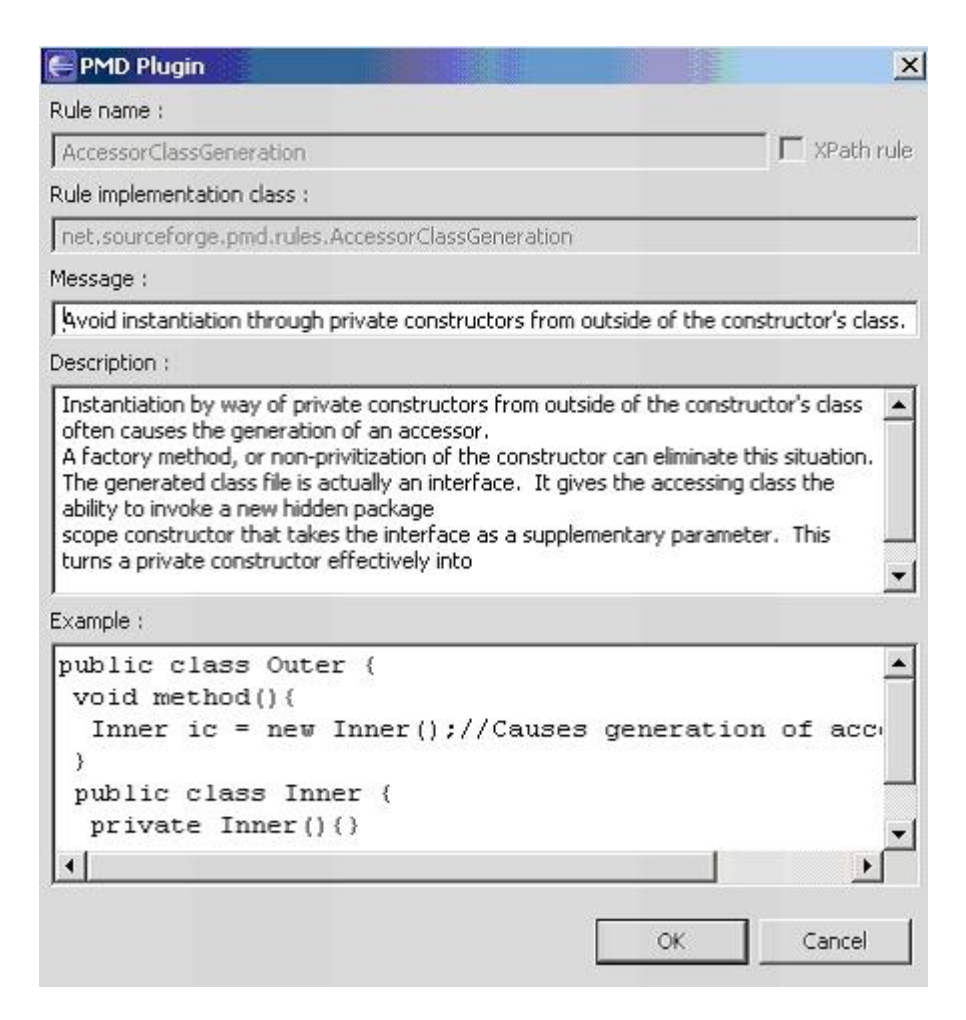
b. En caso de haber creado la regla mediante una expresión XPath activar la casilla de verificación *XPath rule*. En ese caso aparecerá el valor por defecto del campo *Rule Implementation Class*. Y habrá que añadir una propiedad *xpath* en la que añadir el valor de la expresión XPath.




### Modificar una regla

Desde la ventana de preferencias (Window→ Preferences...), pulsamos la línea “PMD→ Rules configuration” que permite gestionar las reglas a aplicar a nuestros proyectos.

1. Seleccionar la regla a modificar.
2. Pulsar el botón “Edit rule…” y modificar los campos mensaje, descripción y/o ejemplo.




**Eliminar una regla**

Desde la ventana de preferencias (Window→ Preferences...), pulsamos la línea “PMD→ Rules configuration” que permite gestionar las reglas a aplicar a nuestros proyectos.

1. Seleccionar la regla a eliminar.
2. Pulsar el botón “Remove rule”.

### 3.1.2. Exportar reglas

En caso de haber creado nuevas reglas, modificado o eliminado alguna de las existentes existe la posibilidad de exportar el conjunto de reglas actual a un nuevo archivo.

Desde la ventana de preferencias (Window→ Preferences...), pulsamos la línea “PMD→ Rules configuration” que permite gestionar las reglas a aplicar a nuestros proyectos.

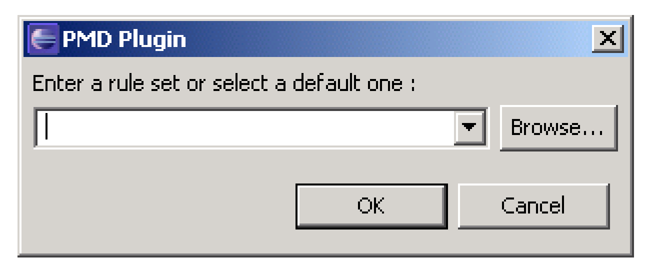
1. Pulsar el botón “Export rule set…”.
2. Guardar el archivo actual de reglas en una carpeta a determinar por el usuario, y dando un nombre

identificativo a dicho archivo para poder recuperarlo a posteriori si se considerase necesario.

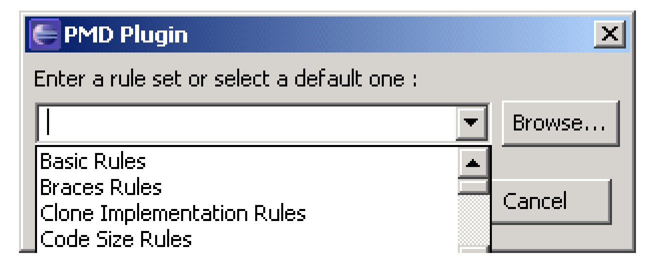
### 3.1.3. Importar reglas

Desde la ventana de preferencias (Window→ Preferences...), pulsamos la línea "PMD→ Rules configuration" que permite gestionar las reglas a aplicar a nuestros proyectos.

1. Pulsar sobre el botón "Clear all". Este paso eliminará todas las reglas previamente definidas. Se pedirá confirmación de la operación. Si la respuesta es afirmativa, se procederá a la supresión de dichas reglas, quedando ahora la pantalla de reglas completamente vacía.
2. Pulsar sobre el botón "Import rule set…". Se puede proceder a importar una serie de reglas ya disponibles y predefinidas por la herramienta o bien cargarlas desde un fichero XML (suministrado en el proceso de instalación de la herramienta).




   a. Para cargar reglas predefinidas, las seleccionamos desde el desplegable y pulsamos OK, realizando este paso tantas veces como conjunto de reglas queramos cargar



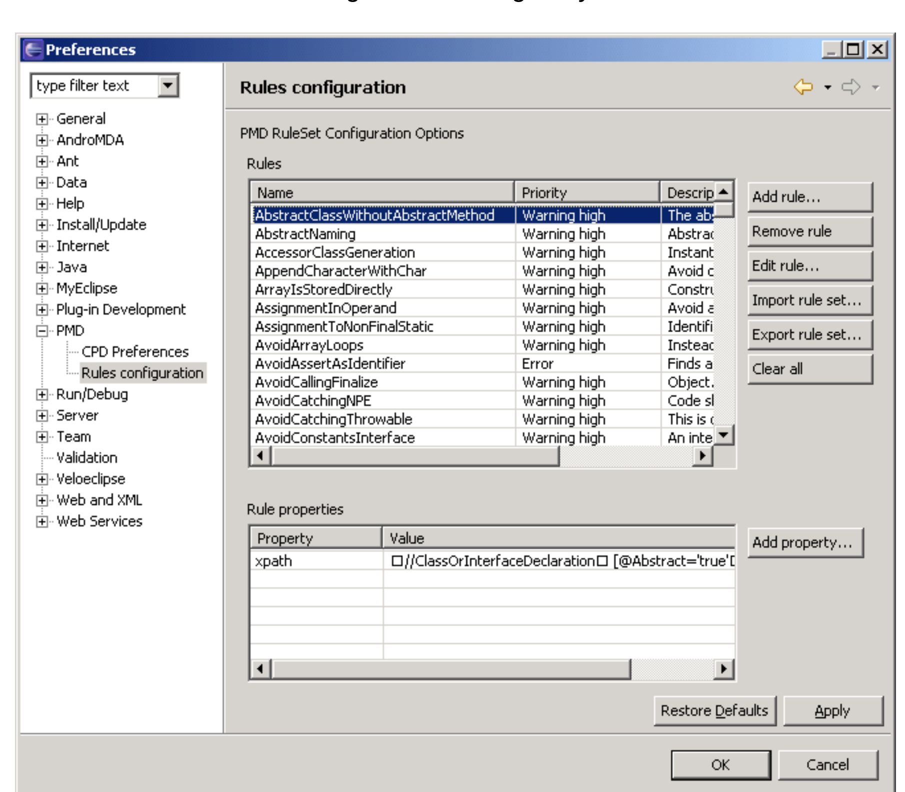

   b. Para cargarlas desde un fichero XML, en nuestro caso el fichero "PMDconfig.xml" elaborado para EJIE, se deberá pulsar sobre el botón "Browse…" y seleccionarlo en el directorio local donde tengamos almacenado este fichero.

Salvo casos excepcionales y justificados, deberemos siempre cargar las reglas del fichero PMDconfig.xml de Ejie. Se puede descargar junto con la carpeta rulesets, desde el SPS de CAC, en:

http://elkarlan.ejie/webguneak/cac/espacio_compartido/default.aspx?RootFolder=%2fwebguneak%2fcac%2fespacio%5fcompartido%2fHerramientas%20de%20desarrollo%2fDescargables%2fReglas%20PMD&FolderCTID=&View=%7bD4F947E6%2dAC1E%2d4C9D%2dAAB7%2dCB2F7A5EDF49%7d

Tanto el fichero PMDconfig.xml como la carpeta rulesets descargados, deben ir al mismo directorio.

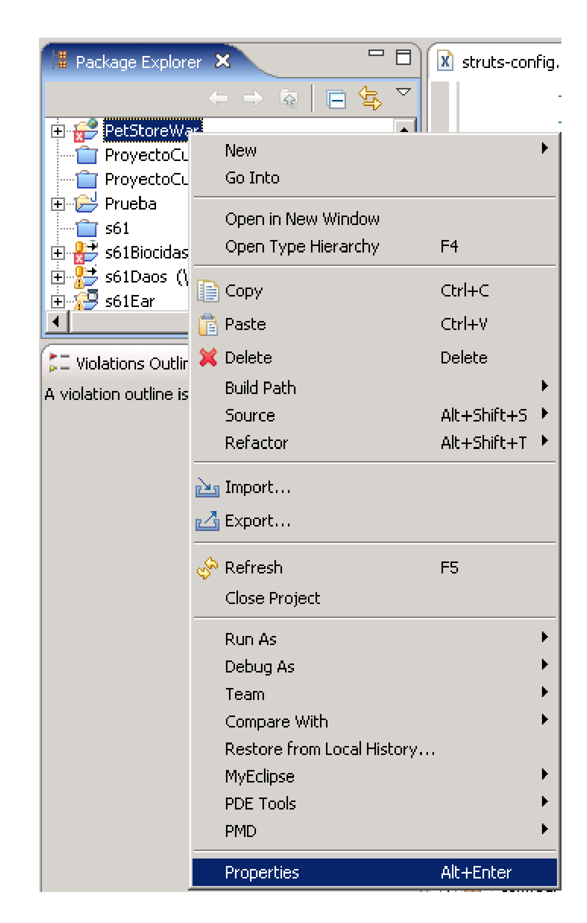
Veremos como la lista de reglas se ha cargado ya con las seleccionadas.



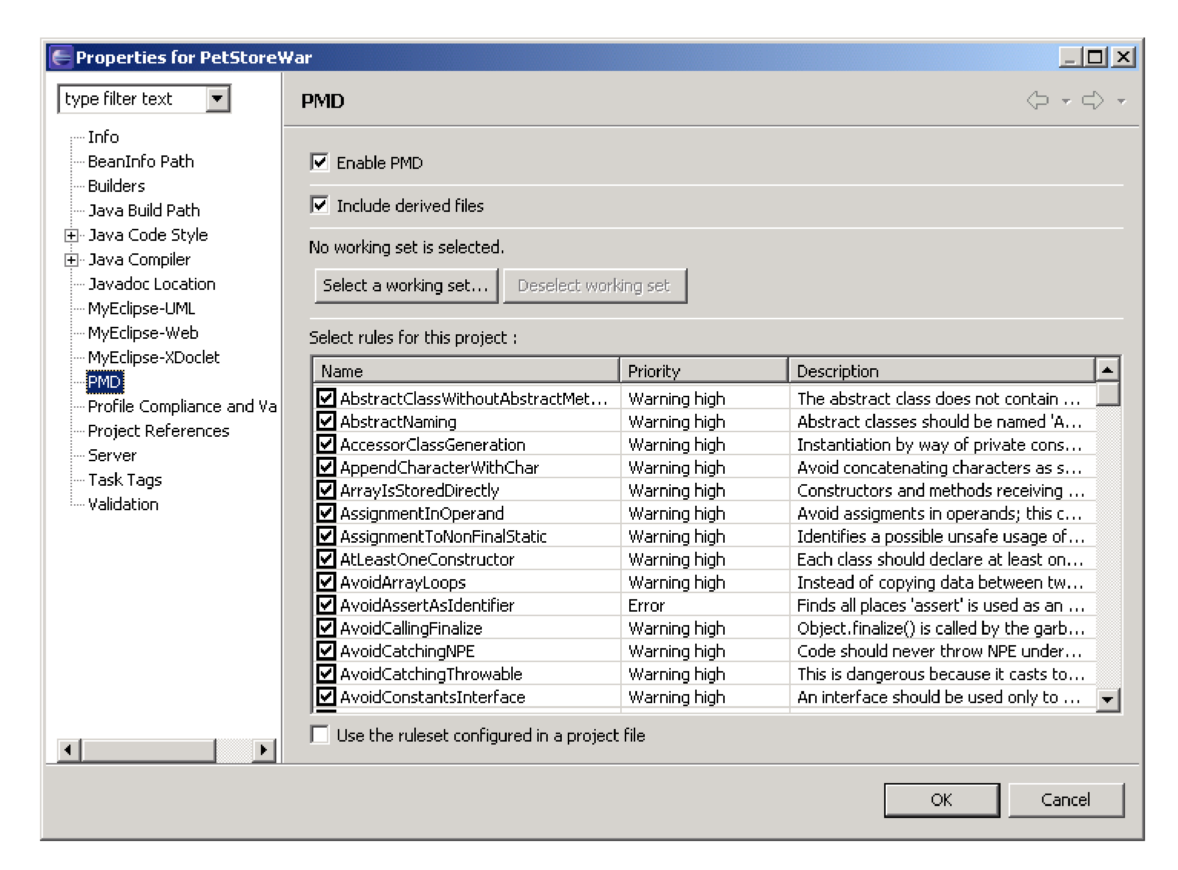

## 3.2 Parametrización de PMD para el proyecto

Para cada proyecto, pueden especificarse diferentes reglas seleccionándolas de las establecidas a nivel general.

Para ello, en Eclipse, en la ventana de “Package Store”, se hace un clic derecho de ratón, y se accede al apartado de “Properties”.




Una vez se ha accedido a este apartado, muestra la siguiente ventana de información:




Desde dicha ventana se permitirá:

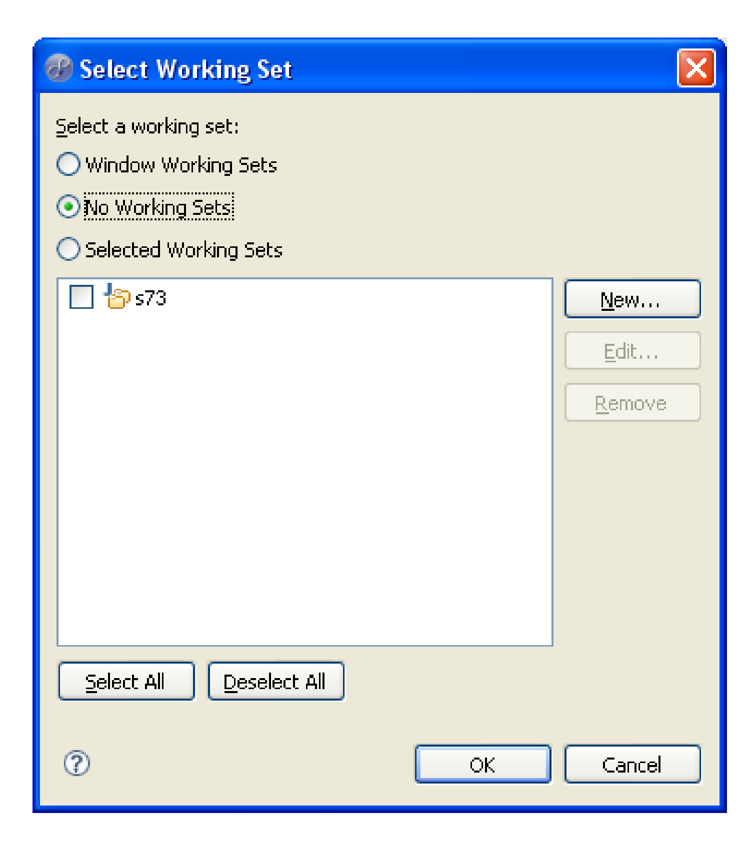
- Habilitar o deshabilitar PMD para el proyecto seleccionado.
- Incluir o no los ficheros derivados.
- Definir los ficheros y subdirectorios que debe analizar PMD (working set)
- Activar o desactivar reglas para este proyecto a través de las casillas de verificación asociadas a cada regla.

Salvo causa justificada, se deberán utilizar las reglas del fichero PMDconfig.xml de EJIE en todos los proyectos. En el apartado Importar reglas se describe el proceso de carga de estas reglas.

### 3.2.1. Definir “Working set”

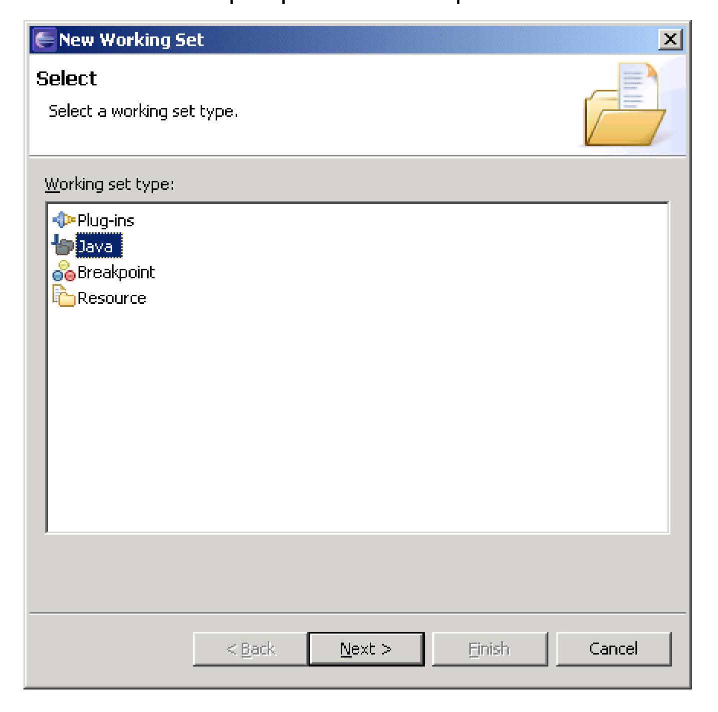
Un “working set” no es más que un conjunto de ficheros y subdirectorios de nuestros proyectos Eclipse. Para definir un “working set”, desde la ventana de propiedades del proyecto (ventana “Package Store”, botón derecho, menú “Properties”) se selecciona la línea de configuración de PMD para el proyecto.

1. Pulsar el botón “Select a working set”



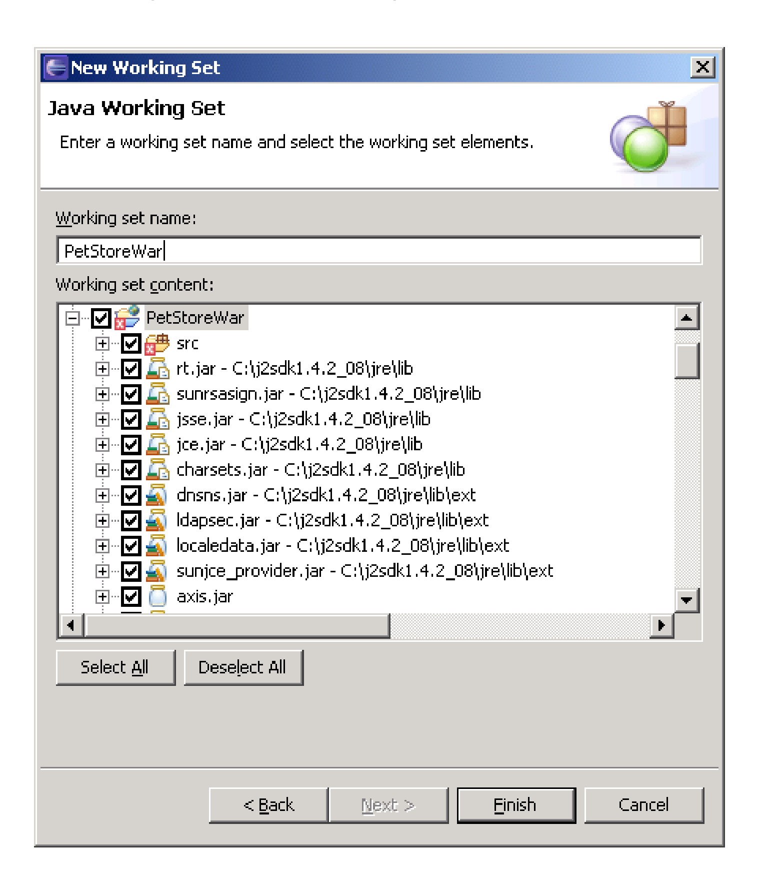

Desde dónde podremos seleccionar alguno creado previamente, o bien crear uno nuevo.

2. Para crear uno nuevo, se pulsará sobre el botón "New…" , el cual nos dará la opción de seleccionar los archivos que queremos incorporar:

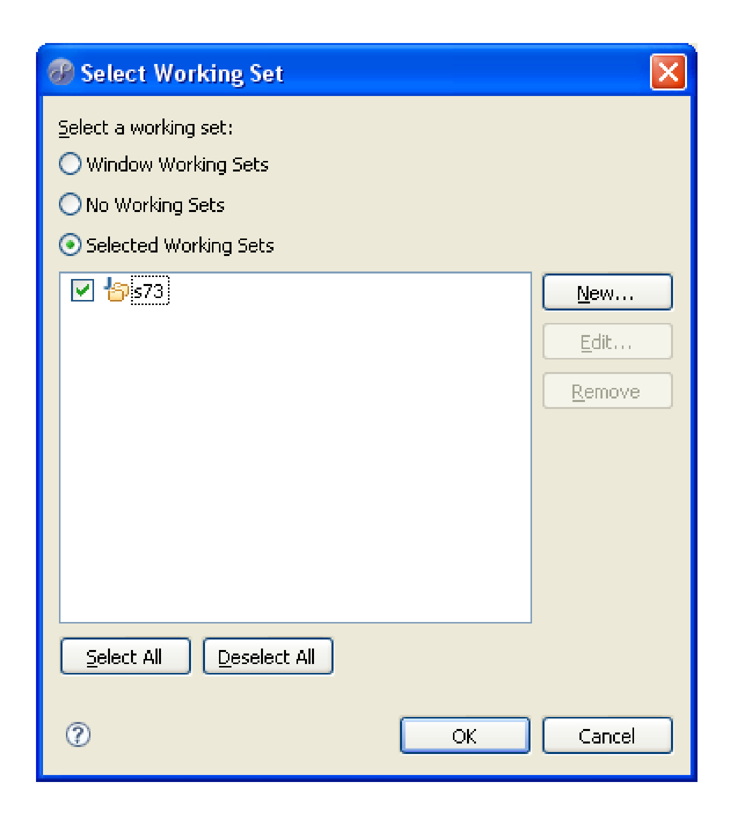


Seleccionamos la opción de Java y pulsamos “Next”, con lo que deberemos marcar también los ficheros de los espacios de trabajo que queremos incorporar, y establecer el nombre que queremos dar al “working set”.

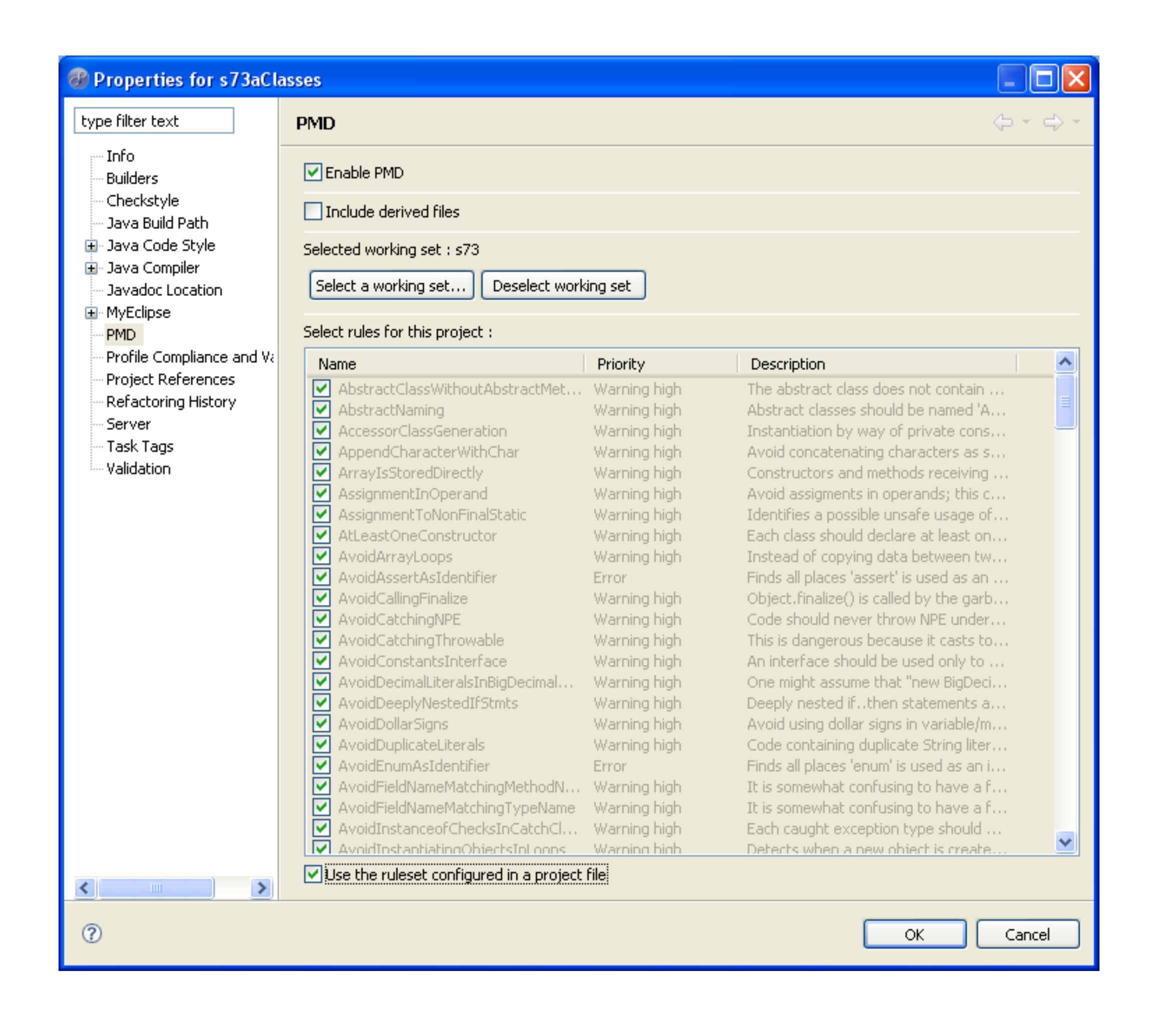
3. Para guardar la selección pulsamos “Finish”




4. Marcamos los “working set” deseados y pulsamos OK



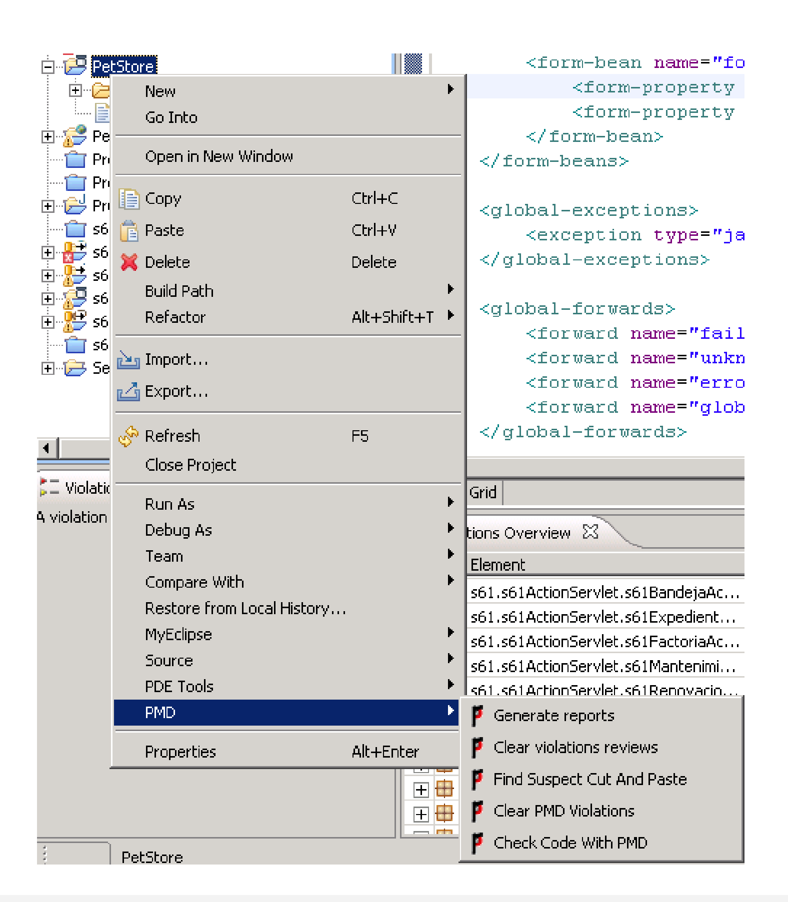

5. Se pueden activar o desactivar las reglas configuradas en el proyecto. Si se activa la casilla “Use the ruleset configured in a Project file”, esto no será posible y se validarán las reglas definidas en el fichero “.ruleset” del proyecto.




Salvo excepción se deberán utilizar las reglas homologadas en EJIE. En el apartado Importar reglas se describe cómo cargar estas reglas.

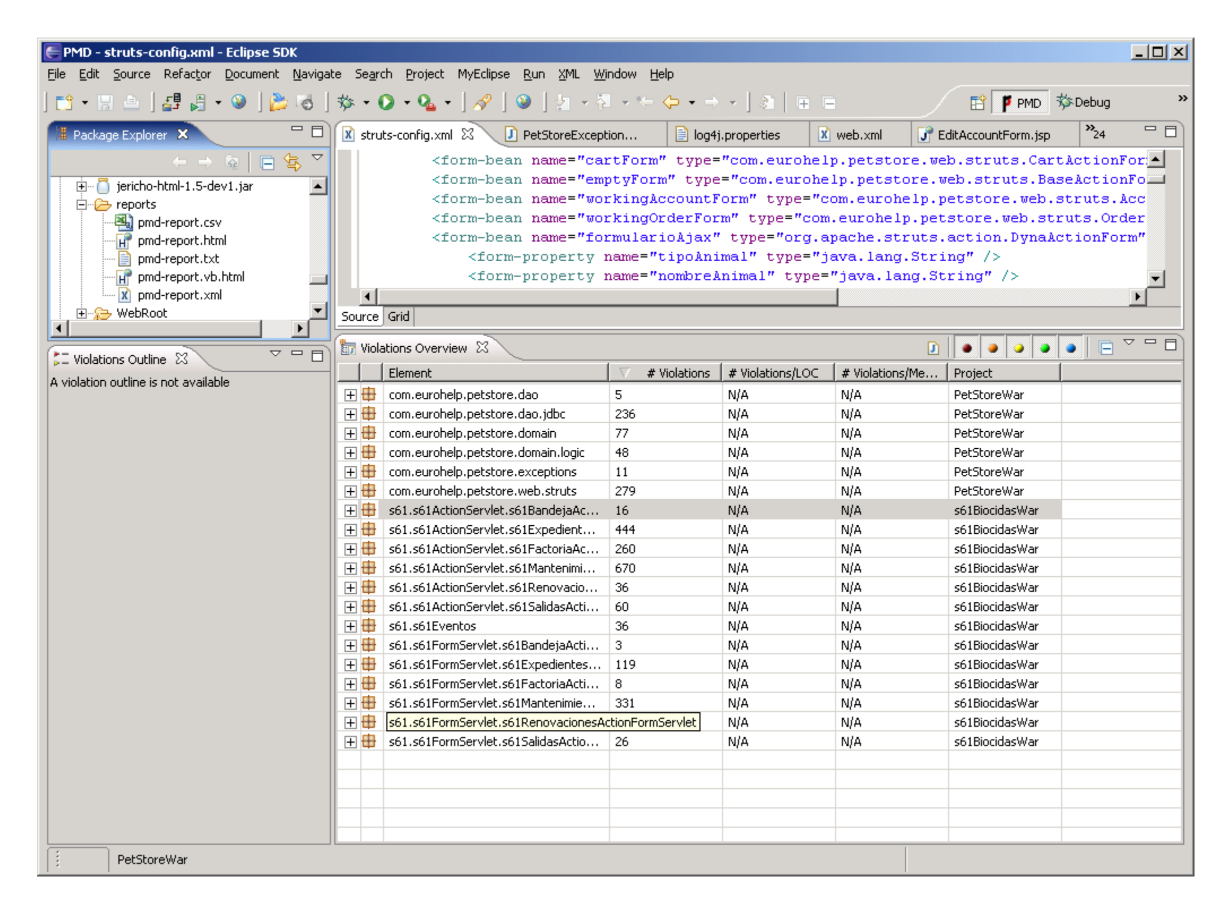
## 3.3 Ejecución de PMD

PMD al estar integrado con Eclipse se arranca al lanzar el IDE. El proceso de arranque, es tan simple como lanzar el ejecutable del eclipse donde se tenga configurado el plug-in del PMD. Para ello, con el botón derecho del ratón, veremos que existe una nueva opción de menú denominada PMD.




## 3.3.1. Generar informes

La opción "**Generate Reports**" permite obtener informes, creando una carpeta denominada "reports", accesible desde la ventana "Package Explorer". Esto genera diferentes tipos de informes consultables sin problema alguno por cualquier herramienta, desde un navegador Web, un editor de texto, un visor de archivos en formato CSV e incluso en formato XML.




Un ejemplo de archivo de informe de salida puede ser el que a continuación se detalla:

# PMD report
# Problems found

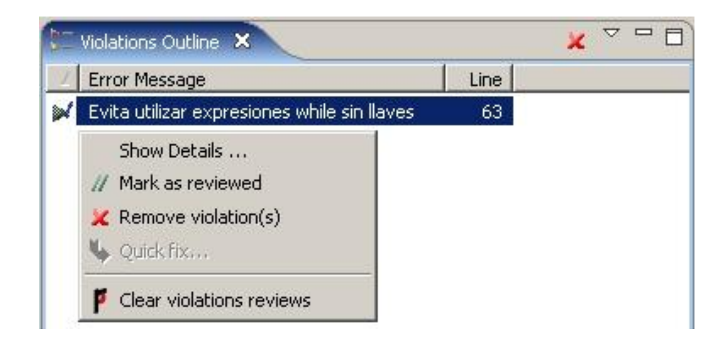
| # | File | Line | Problem |
|---|---|---|---|
| 1 | src/com/eurohelp/petstore/dao/ItemDao.java | 10 | Avoid modifiers which are implied by the context |
| 2 | src/com/eurohelp/petstore/dao/Sequence.java | 5 | Classes implementing Serializable should set a serialVersionUID |
| 3 | src/com/eurohelp/petstore/dao/Sequence.java | 14 | Document empty constructor |
| 4 | src/com/eurohelp/petstore/dao/Sequence.java | 25 | Parameter 'name' is not assigned and could be declared final |
| 5 | src/com/eurohelp/petstore/dao/Sequence.java | 28 | Parameter 'nextId' is not assigned and could be declared final |
| 6 | src/com/eurohelp/petstore/dao/jdbc/AccountDaoImpl.java | 18 | Each class should declare at least one constructor |
| 7 | src/com/eurohelp/petstore/dao/jdbc/AccountDaoImpl.java | 20 | Found non-transient, non-static member. Please mark as transient or provide accessors. |
| 8 | src/com/eurohelp/petstore/dao/jdbc/AccountDaoImpl.java | 22 | Parameter 'jdbcTemplate' is not assigned and could be declared final |
| 9 | src/com/eurohelp/petstore/dao/jdbc/AccountDaoImpl.java | 26 | Parameter 'username' is not assigned and could be declared final |
| 10 | src/com/eurohelp/petstore/dao/jdbc/AccountDaoImpl.java | 53 | Local variable could be declared final |
| 11 | src/com/eurohelp/petstore/dao/jdbc/AccountDaoImpl.java | 53 | StringBuffer constructor is initialized with size 16, but has at least 494 characters appended |
| 12 | src/com/eurohelp/petstore/dao/jdbc/AccountDaoImpl.java | 56 | StringBuffer.append is called 32 consecutive times with literal Strings. Use a single append with a single String. |
| 13 | src/com/eurohelp/petstore/dao/jdbc/AccountDaoImpl.java | 78 | The String literal " WHERE " appears 5 times in this file; the first occurrence is on line 78 |

Los resultados se muestran en cuatro columnas:

- La primera columna asocia un número incremental a cada uno de los problemas encontrados tras realizar la verificación del código de un proyecto (nos permite conocer el número total de problemas).
- La segunda (*File*) muestra la clase para la que se ha detectado el problema (puede haber más de un aviso por clase como se muestra en el informe anterior).
- La tercera (*Line*) muestra el número de línea de código en la que se ha detectado dicho problema.
- La última columna (*Problem*) muestra la descripción del problema que ha causado el aviso de PMD (descripción relacionada con una clase específica dentro de un conjunto concreto de reglas de validación habilitado para realizar la verificación del código).

### 3.3.2. Eliminar incidencias revisadas (Uso de NOPMD)

Existe la posibilidad de acceder a la vista *Violations Outline* donde se describen los errores encontrados para una clase concreta. Seleccionando una descripción podrá marcarse un error como revisado a través de la opción “Mark as reviewed” lo que generará un comentario en la línea de código donde se produjo dicho error.




Sentencia que produjo el error:

```
while (cartItems.hasNext())
```

Sentencia con la marca (comentario) generada:

```
while (cartItems.hasNext())  // NOPMD by lmgurrea on 3/08/07 12:15
```

**NOTA:** En realidad esto sólo se debe utilizar para falsos positivos que pueda devolver PMD. Además hay que indicar un motivo por el cuál se 'marca como revisado' a la dcha. del nombre de un responsable y la fecha. En lugar de utilizar la opción marked as review, es mejor utilizar un template de MyEclipse del tipo:

```
//NOPMD @responsable=${user} @fecha=${date} @motivo=${cursor}
```

Resulta útil hacer un template en:
Window>Preferences>Java>Editor>Templates
Con nombre NOPMD

De forma que cuando escribamos a la derecha de la línea de código que dé el falso positivo 'NOPMD' y pulsemos ctrl.+ space se nos escribirá automáticamente lo siguiente:

```
//NOPMD @responsable='Usuario responsable' @fecha='Fecha' @motivo=
```

Y el cursor se nos quedará detrás de 'motivo=', listo para indicar la razón del uso de NOPMD. Siempre hay que indicar el motivo por el cual no vamos a tener en consideración ese aviso de PMD.

La opción "**Clear Violations Reviews**", permite la supresión de las marcas creadas para las incidencias ya revisadas. Por lo tanto eliminará los comentarios añadidos anteriormente.

### 3.3.3. Buscar código redundante (CPD)

La opción "**Find Suspect Cut And Paste**" permite la ejecución del plug-in CPD, para la búsqueda de código redundante en el proyecto. Genera un informe, que se ubicará dentro de la carpeta "reports", pudiendo ser éste un ejemplo de salida:

```
=====================================================================
Found a 26 line (250 tokens) duplication in the following files:
Starting at line 337 of
C:\Proyectos\s61Workspace\PetStoreWar\src\com\eurohelp\petstore\dao\jdbc\OrderDaoImpl.java
```

```
Starting at line 430 of
C:\Proyectos\s61Workspace\PetStoreWar\src\com\eurohelp\petstore\dao\jdbc\OrderDaoImpl.java
                              order = new Order();

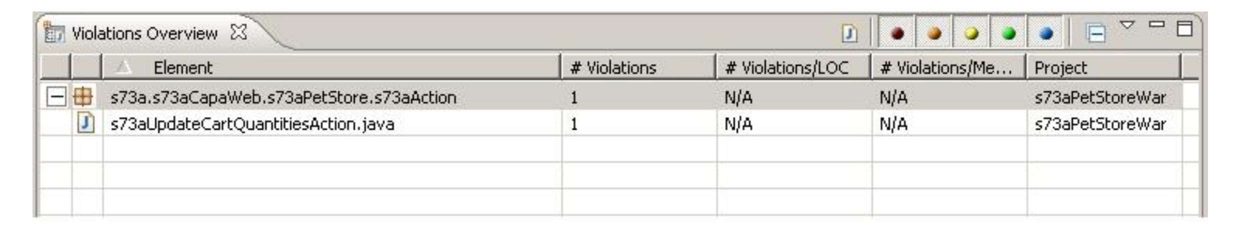
                              order.setBillAddress1(rs.getString("billaddr1"));
                              order.setBillAddress2(rs.getString("billaddr2"));
                              order.setBillCity(rs.getString("billcity"));
                              order.setBillCountry(rs.getString("billcountry"));
                              order.setBillState(rs.getString("billstate"));
                              order.setBillToFirstName(rs.getString("billtofirstname"));
                              order.setBillToLastName(rs.getString("billtolastname"));
                              order.setBillZip(rs.getString("billzip"));
                              order.setShipAddress1(rs.getString("shipaddr1"));
                              order.setShipAddress2(rs.getString("shipaddr2"));
                              order.setShipCity(rs.getString("shipcity"));
                              order.setShipCountry(rs.getString("shipcountry"));
                              order.setShipState(rs.getString("shipstate"));
                              order.setShipToFirstName(rs.getString("shiptofirstname"));
                              order.setShipToLastName(rs.getString("shiptolastname"));
                              order.setShipZip(rs.getString("shipzip"));
                              order.setCardType(rs.getString("cardtype"));
                              order.setCourier(rs.getString("courier"));
                              order.setCreditCard(rs.getString("creditcard"));
                              order.setExpiryDate(rs.getString("exprdate"));
                              order.setLocale(rs.getString("locale"));
                              order.setOrderDate(rs.getDate("orderdate"));

                              order.setTotalPrice(rs.getDouble("totalprice"));
=====================================================================
```

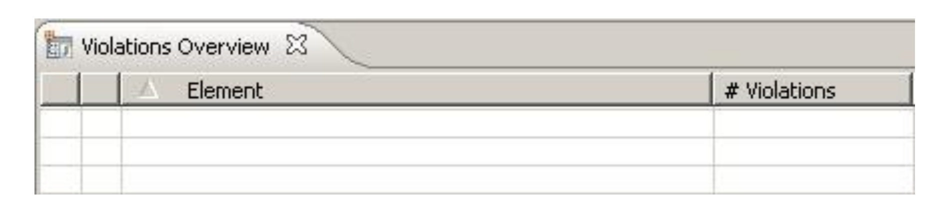
El informe generado muestra el código que aparece duplicado (puede estar duplicado dentro de la misma clase o en varias clases distintas) e indica la clase a la que pertenece dicho código duplicado y a partir de la línea en la que aparece. En este caso concreto el código que se repite (26 líneas) reside en la misma clase y está implementado a partir de la línea 337 y duplicado a partir de la línea 430 o viceversa.

### 3.3.4. Eliminar incidencias

Las incidencias que detecta PMD aparecen reflejadas en la vista “Violations Overview”:

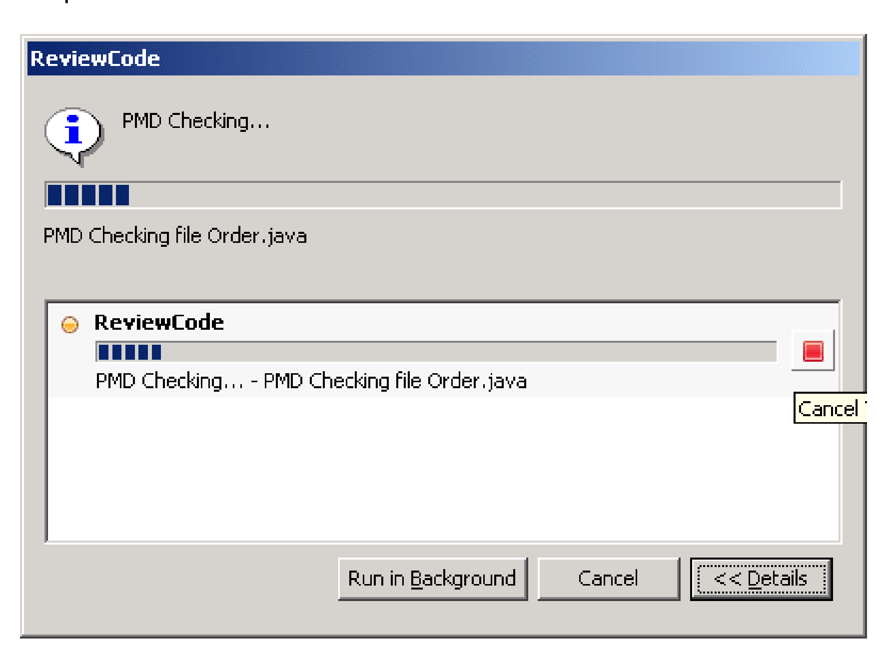



La opción “**Clear PMD Violations**”, permite limpiar la lista de incidencias de dicha vista:




### 3.3.5. Chequear código

La opción “**Check Code with PMD**”, permite revisar el código bajo PMD, permitiendo incluso que dicho chequeo pueda efectuarse en modo Background y poder seguir trabajando sin la necesidad de esperar a que dicho chequeo finalice.



# 4 Utilidad práctica

PMD proporciona un amplio conjunto de reglas de verificación de código en base a una serie de pautas para la aplicación de buenas técnicas de implementación. Normalmente será suficiente con los conjuntos de reglas aportados por la propia herramienta pero existe la posibilidad de creación de reglas de validación personalizadas.

## 4.1 Aportaciones de PMD

A continuación se muestra el listado de los conjuntos de reglas aportadas por PMD así como una breve descripción de su utilidad. Cada uno de estos conjuntos incluye una colección de clases que realizan las verificaciones específicas. Los enlaces aportados muestran la funcionalidad concreta de cada una de estas clases:

- Basic JSF rules: Conjunto de reglas con pautas básicas de JSF.
- Basic JSP rules: Conjunto de reglas con pautas básicas de JSP.
- Basic Rules: Colección de buenas prácticas que todo el mundo debería seguir.
- Braces Rules: Colección de reglas para el uso de llaves.
- Clone Implementation Rules: Conjunto de reglas para encontrar usos cuestionables del método *clone()*.
- Code Size Rules: Reglas relacionadas con el tamaño del código.
- Controversial Rules: Conjunto de reglas que pueden ser consideradas, de alguna manera, controvertidas. Por ello se han agrupado en esta división.
- Coupling Rules: Conjuntos de reglas que buscan instancias de uniones inapropiadas entre objetos y paquetes.
- Design Rules: Reglas que buscan diseños cuestionables.
- Finalizer Rules: Conjunto de reglas para encontrar usos cuestionables del método *finalize()*.
- Import Statement Rules: Reglas que buscan problemas con las sentencias *import*.
- J2EE Rules: Reglas para J2EE.
- JavaBean Rules: Conjunto de reglas para JavaBeans.
- JUnit Rules: Reglas que buscan errores que pueden producirse en los testes de JUnit.
- Jakarta Commons Logging Rules: Contiene un conjunto de reglas que busca usos cuestionables de este framework.
- Java Logging Rules: Reglas que buscan usos cuestionables de Log.
- Migration Rules: Contienen reglas para la migración de una versión de JDK a otra. Es necesario usar los conjuntos de reglas específicos para tales fines que se describen a continuación.
- Migration13: Reglas para la migración a JDK 1.3.
- Migration14: Reglas para la migración a JDK 1.4.
- Migration15: Reglas para la migración a JDK 1.5.
- MigratingToJava4: Reglas para la migración a JDK 1.5.
- Naming Rules: Colección de reglas sobre el uso de nombres (demasiado cortos, demasiado largos, etc.).
- Optimization Rules: Reglas para asegurar un mayor rendimiento.
- Strict Exception Rules: Proporcionan pautas estrictas para la propagación y recogida de excepciones.
- String and StringBuffer Rules: Reglas que tratan diversos que pueden aparecer a la hora de utilizar las clases String o StringBuffer.
- Security Code Guidelines: Estas reglas comprueban las pautas de seguridad aportadas por Sun y

publicadas en http://java.sun.com/security/seccodeguide.html#gcg.
- Type Resolution Rules: Conjunto de reglas que verifican el uso de tipos de datos apropiados en cada caso, etc.
- Unused Code Rules: Reglas para encontrar código no utilizado (variables locales, métodos privados, etc.).

## 4.2 Aportaciones personalizadas

A pesar de que los conjuntos de reglas de verificación aportados son extensos siempre existe la posibilidad de modificar el funcionamiento de ciertas reglas de validación para ajustarlas a unas necesidades concretas o bien crear nuevas reglas.

Por ejemplo, puede resultar interesante:

- Comprobar si se han creado objetos de tipo *Thread* a través de instrucciones *Thread t = new Thread();*.
- Prevenir la probabilidad de encontrar conexiones no cerradas tras comprobar la existencia de objetos de tipo *Connection*.
- Prevenir ante el uso de objetos del tipo *PrintStrackTrace*.
- Evitar el uso de expresiones regulares en la implementación de las clases para facilitar su posterior mantenimiento.
- Evitar el uso de instrucciones que pueden llegar a complicar su legibilidad como en el caso de la expresión condicional *if (expresion Boolean ? operación verdadera : operación falsa)*.
- Etc.

Los siguientes apartados muestran los pasos a seguir para crear una regla de validación personalizada y su, más que probable, posterior uso desde Eclipse.

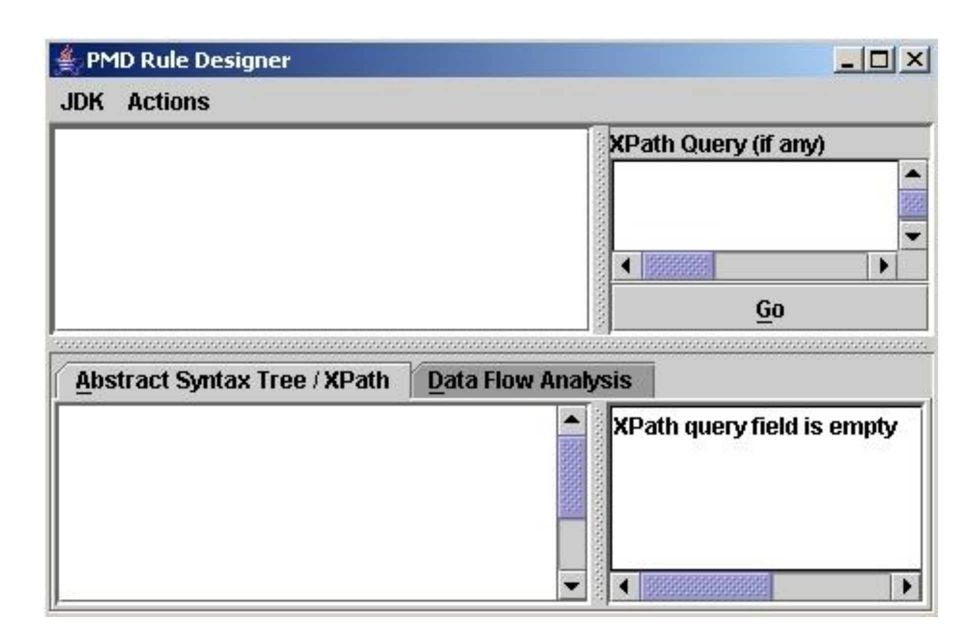
## 4.3 Creación de reglas de validación personalizadas

Es necesario explicar brevemente cómo funciona PMD. PMD no analiza el código fuente de manera directa. PMD “parsea” dicho código para generar un elemento AST (Abstract Syntax Tree).

PMD (http://pmd.sourceforge.net/) nos ofrece una herramienta para transformar código fuente en un elemento AST. En primer lugar, descargamos y descomprimimos el archivo *pmd-bin-4.0.zip*. Se nos creará la siguiente estructura de directorios:




Para acceder a dicha herramienta se ejecutará el archivo *designer.bat*, el cual se encuentra dentro del directorio *bin*. El aspecto que presenta esta herramienta es el siguiente:




Como un ejemplo sencillo de regla personalizada se creará una regla que detecte los bucles *while* que no rodeen su contenido con llaves "{}".

En primer lugar vamos a ver la representación AST de un código incorrecto para, posteriormente, compararlo con la representación AST del código fuente correcto, es decir, utilizando las llaves "{}". Para ello:

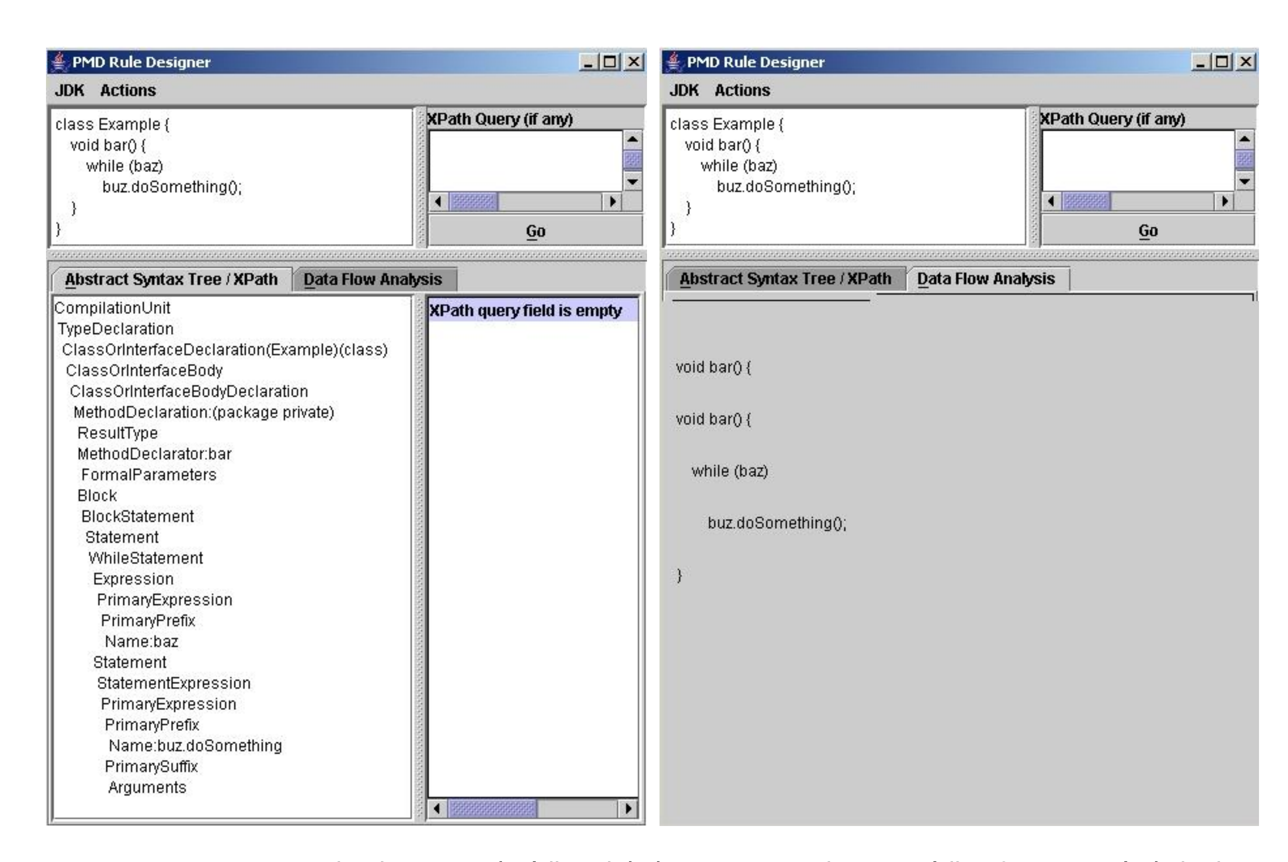
1. Se introducirá el siguiente código fuente en el área superior izquierda de la herramienta *designer.bat:*

```
class Example {
   void bar() {
      while (baz)
         buz.doSomething();
   }
}
```

2. Se seleccionará el JDK adecuado desde la opción correspondiente de la barra de herramientas:




3. Se pulsará el botón "Go" obteniendo así en la pestaña *Abstract Syntax Tree / XPath* el elemento AST correspondiente:




Por tanto, como se puede observar, el código del elemento AST de este código fuente será el siguiente:

```
CompilationUnit
 TypeDeclaration
  ClassOrInterfaceDeclaration(Example)(class)
   ClassOrInterfaceBody
    ClassOrInterfaceBodyDeclaration
     MethodDeclaration:(package private)
      ResultType
      MethodDeclarator:bar
       FormalParameters
      Block
       BlockStatement
        Statement
         WhileStatement
          Expression
           PrimaryExpression
            PrimaryPrefix
             Name:baz
          Statement
           StatementExpression
            PrimaryExpression
             PrimaryPrefix
              Name:buz.doSomething
             PrimarySuffix
              Arguments
```

Nos interesa el código del elemento AST relacionado, concretamente, con el bucle *while*:

```
WhileStatement
 Expression

 Statement
  StatementExpression
```

Ahora vemos la representación AST de un código correcto. Para ello ejecutamos el nuevo código con la herramienta *designer.bat*. Podemos observar que ahora sí se han incluido las llaves “{}” rodeando el código contenido en el bucle *while*:

```
class Example {
    void bar() {
        while (baz) {
            buz.doSomething();
        }
    }
}
```

El elemento AST obtenido es:

```
CompilationUnit
 TypeDeclaration
  ClassOrInterfaceDeclaration(Example)(class)
   ClassOrInterfaceBody
    ClassOrInterfaceBodyDeclaration
     MethodDeclaration:(package private)
      ResultType
      MethodDeclarator:bar
       FormalParameters
      Block
       BlockStatement
        Statement
         WhileStatement
          Expression
           PrimaryExpression
            PrimaryPrefix
             Name:baz
          Statement
           Block
            BlockStatement
             Statement
              StatementExpression
               PrimaryExpression
                PrimaryPrefix
                 Name:buz.doSomething
                PrimarySuffix
                 Arguments
```

Y el código del elemento AST relacionado con el bucle *while*:

```
WhileStatement
```

```
    Expression

    Statement
     Block
      BlockStatement
       Statement
        StatementExpression
```

De esta manera y según nuestro ejemplo, para que un bucle *while* sea considerado correcto, en la transformación de su código fuente a un elemento AST se deberá obtener un elemento *WhileStatement* que contenga un elemento *Statement*. Y dentro de éste, a su vez, se deberá encontrar un elemento *Block* (el cual hace referencia a un bloque de código contenido entre llaves "{}" que es lo que se desea comprobar).

Como ya se ha visto, existen dos maneras de crear una nueva regla:

- Usando Java.
- Usando una expresión XPath.

### 4.3.1. Implementación de una regla personalizada a través de una clase Java

Se deberán seguir los siguientes pasos:

1. En primer lugar se implementará la clase que contiene la regla de validación:

```
import net.sourceforge.pmd.*;
import net.sourceforge.pmd.ast.*;

public class DetectarBucleIncorrecto extends AbstractRule {
   public Object visit(ASTWhileStatement node, Object data) {
      SimpleNode firstStmt = (SimpleNode)node.jjtGetChild(1);
      if ( !hasBlockAsFirstChild(firstStmt) ) {
         addViolation(data, node);
      }
      return super.visit(node,data);
   }
   private boolean hasBlockAsFirstChild(SimpleNode node) {
      return ( node.jjtGetNumChildren() != 0 && (node.jjtGetChild(0) instanceof ASTBlock) );
   }
}
```

Se puede observar que la clase *DetectarBucleIncorrecto* implementada hereda de la clase *AbstractRule*. El método *visit* será llamado cuando se encuentre un elemento *ASTWhileStatement*.

La siguiente instrucción recoge el segundo hijo del elemento *ASTWhileStatement*:

```
SimpleNode firstStmt = (SimpleNode)node.jjtGetChild(1);
```

Y se comprueba si este nodo, a su vez, tiene o no como primer hijo un elemento de tipo ASTBlock:

```
return ( node.jjtGetNumChildren() != 0 && (node.jjtGetChild(0) instanceof ASTBlock) );
```




En caso negativo, se producirá una violación de la regla considerada:

```
addViolation(data, node);
```

2. Una vez implementada la clase, necesitamos añadir esta regla a un archivo XML de definición de un conjunto de reglas, por ejemplo *mycustomrules.xml*.

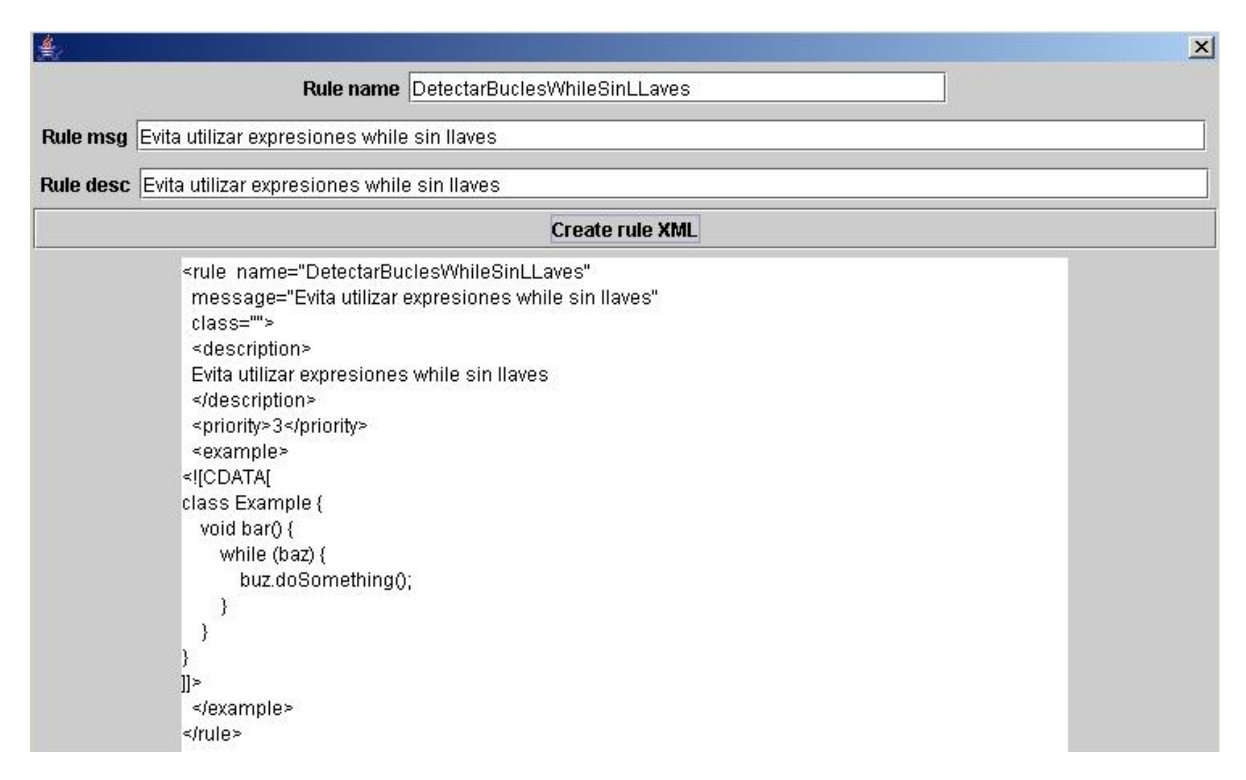
La regla creada deberá presentar los siguientes elementos y atributos:

- **name**: el nombre elegido para reconocer la regla.
- **message**: el mensaje que debe mostrar la regla cuando se detecte una violación.
- **class**: la clase que contiene la regla de validación, puede estar en cualquier sitio.
- **description**: la descripción de la regla.
- **example**: un pequeño código que muestre un ejemplo de violación de la regla.

Nos puede servir de ayuda, para crear la regla, la opción de menú *Create rule XML* incluida en la herramienta *designer.bat*:




Introduciendo el nombre de la regla, el mensaje y la descripción y pulsando sobre el botón “Create rule XML” nos mostrará el esqueleto del elemento *rule* a introducir en el archivo XML de definición del conjunto de reglas.




Será necesario añadir la clase que contiene la regla de validación dentro del elemento *class* y modificar el

ejemplo para que corresponda con uno que viole la regla que está siendo definida.

El archivo XML de definición del conjunto de reglas, incorporando únicamente esta regla, puede presentar el siguiente aspecto:

```
<?xml version="1.0"?>
<ruleset name="My custom rules"
    xmlns="http://pmd.sf.net/ruleset/1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://pmd.sf.net/ruleset/1.0.0 http://pmd.sf.net/ruleset_xml_schema.xsd"
    xsi:noNamespaceSchemaLocation="http://pmd.sf.net/ruleset_xml_schema.xsd">
    <rule name="DetectarBuclesWhileSinLLaves"
    message="Evita utilizar expresiones while sin llaves"
    class="DetectarBucleIncorrecto">
      <description>
      Evita utilizar expresiones while sin llaves
      </description>

      <priority>3</priority>

      <example>
<![CDATA[
    public void doSomething() {
      while (true)
        x++;
    }
]]>
      </example>
    </rule>
</ruleset>
```

#### 4.3.2. Implementación de una regla personalizada con una expresión XPath

La instrucción *xpath* que define la regla que estamos implementando es el siguiente:

```
//WhileStatement[not(Statement/Block)]
```

Este tipo de implementación también se basa en el código AST generado. Pero en este caso, no es necesaria implementar una clase Java.

Pero, como en el caso anterior, sí es necesario añadir esta regla a un archivo XML de definición de un conjunto de reglas (*mycustomrules.xml*) añadiendo un elemento *property* de nombre *xpath* cuyo valor sea la instrucción *xpath* que hemos definido antes y modificando el atributo *class* con el valor de una clase de la distribución de PMD **net.sourceforge.pmd.rules.XPathRule**.

```
<?xml version="1.0"?>
<ruleset name="My custom rules"
    xmlns="http://pmd.sf.net/ruleset/1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://pmd.sf.net/ruleset/1.0.0 http://pmd.sf.net/ruleset_xml_schema.xsd"
    xsi:noNamespaceSchemaLocation="http://pmd.sf.net/ruleset_xml_schema.xsd">
    <rule name="DetectarBuclesWhileSinLLavesXPath"
```

```
      message="Evita utilizar expresiones while sin llaves"
      class="net.sourceforge.pmd.rules.XPathRule">
        <description>
        Evita utilizar expresiones while sin llaves
        </description>

        <priority>3</priority>

        <properties>
          <property name="xpath">
           <value>
           //WhileStatement[not(Statement/Block)]
           </value>
         </property>
       </properties>

        <example>
  <![CDATA[
      public void doSomething() {
        while (true)
          x++;
      }
  ]]>
        </example>
      </rule>
  </ruleset>
```

### 4.3.3. Cómo usar un conjunto de reglas personalizadas con Eclipse

PMD busca los archivos de conjuntos de reglas en el classpath definido para el plug-in. Por tanto, para usar un conjunto de reglas personalizadas, deberán estar situadas en el classpath del plug-in PMD.

La manera de hacer esto con Eclipse es empaquetar los conjuntos de reglas en un fragmento del plug-in. Un fragmento es una extensión del plug-in y todas las clases y los archivos de recursos que contenga son automáticamente añadidos al classpath principal del plug-in.

Los pasos para añadir un conjunto de reglas personalizado son los siguientes:

1. Crear un fragmento de *Plug-in fragment* como una extensión del plug-in PMD.
2. Implementar las reglas.
3. Crear uno o más conjuntos de reglas.
4. Empaquetar el fragmento.
5. Instalar el fragmento.

**Crear un fragmento de plug-in como una extensión del plug-in PMD**

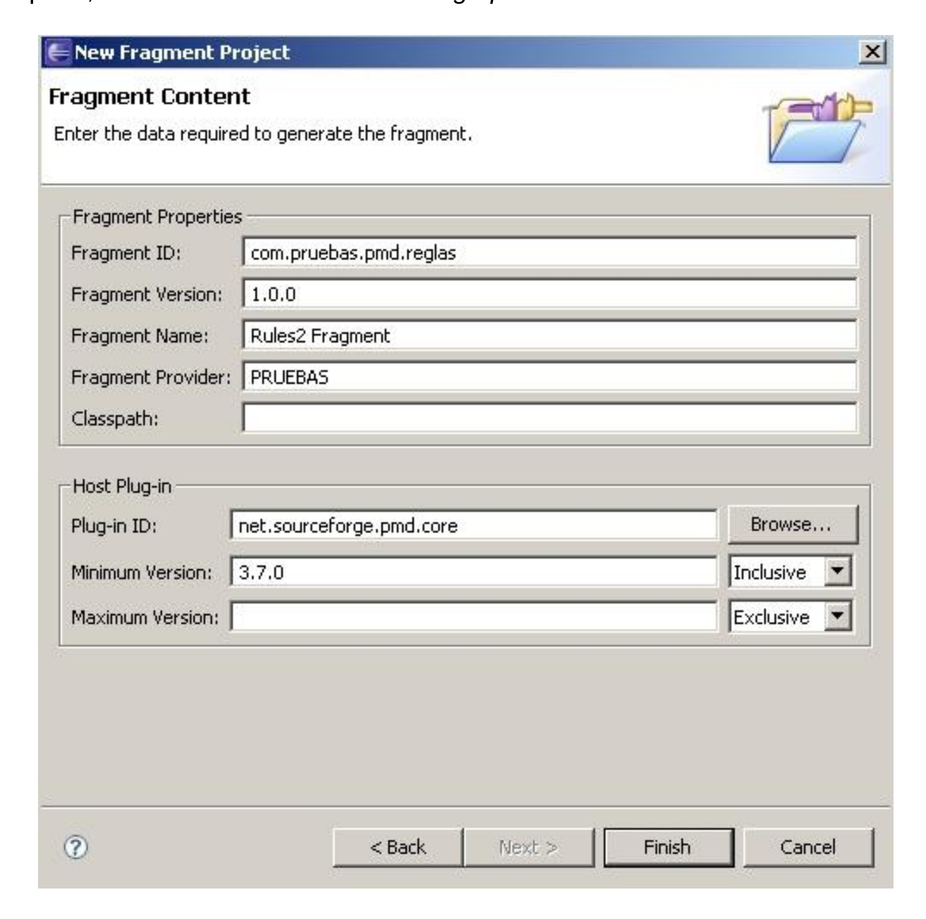
El primer paso es configurar el entorno de desarollo del plug-in. Para ello nos dirigimos a *Window* >> *Preferences* >> *Plug-in development* >> *Target platform* y pulsamos el botón “Select All” y a continuación el botón “OK” para cerrar la ventana.

El segundo paso el crear el proyecto para el fragmento:

Elegimos la opción del menú *File* >> *New Project…* y seleccionamos dentro de la categoría *Plug-in development* el elemento *Fragment Project*. A continuación, pulsamos el botón “Next”.

Damos un nombre al proyecto (un nombre de un proyecto Plug-in normalmente se escribe como el nombre de un paquete, por ejemplo *com.foo.pmd.rules*).

En la siguiente ventana seleccionamos el *Fragment ID*, por defecto el mismo nombre del proyecto. También seleccionamos el *Plug-in ID* el cual debe hacer referencia al ID de un plug-in existente para el workspace, en nuestro caso *net.sourceforge.pmd.core*.




Una vez creado un proyecto *Plug-in Fragment* la estructura obtenida será similar a la siguiente:



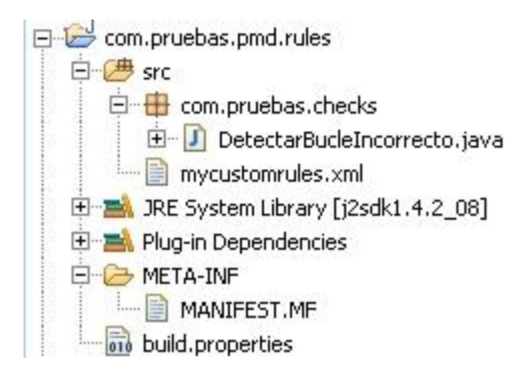

**Implementar las reglas**

Ya hemos visto en un apartado anterior cómo implementar reglas personalizadas. Vamos a hacer uso en

este ejemplo de la clase anteriormente implementada *DetectarBucleIncorrecto.java*. Simplemente es necesario indicar que ésta u otras clases implementadas deberán colocarse en el directorio *src*.

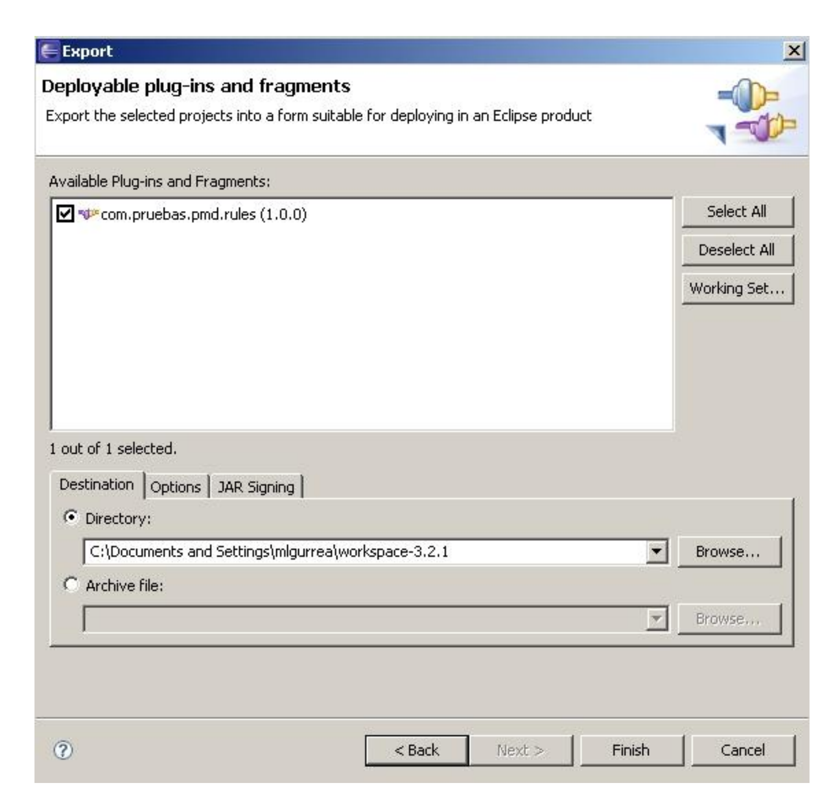
**Crear uno o más conjuntos de reglas**

También hemos visto anteriormente este apartado, por tanto sólo indicar que se deberá colocar el archivo XML de definición de un conjunto de reglas, en este caso el que ya hemos utilizado *mycustomrules.xml* en el directorio source (*src*).

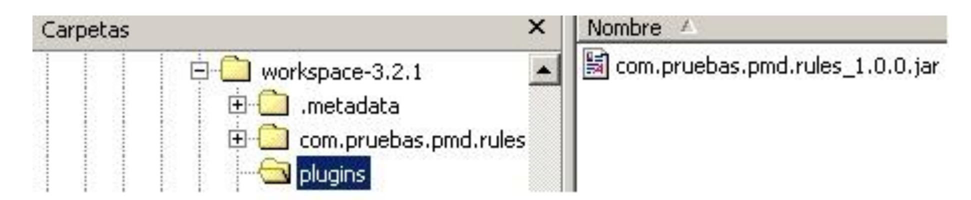



**Empaquetar el fragmento**

También será necesario, antes de desplegar el fragmento, empaquetarlo. Para ello exportaremos el proyecto con la opción *Export…* a la que accederemos situándonos sobre el proyecto *Plug-in Fragment* creado y pulsando el botón derecho del ratón. Tras esto, seleccionaremos *Deployable plug-ins and fragments*. El siguiente paso será seleccionar el plug-in para el que se realiza la extensión y se especificará el directorio de destino.

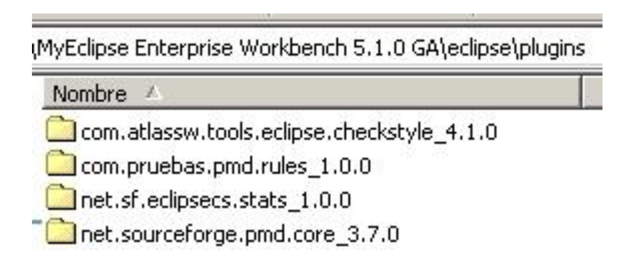


Podremos observar que se nos habrá creado en el directorio de destino una nueva carpeta *plugins* en la que aparecerá el archivo *.jar* creado, en nuestro caso *com.pruebas.pmd.rules_1.0.0.jar*.

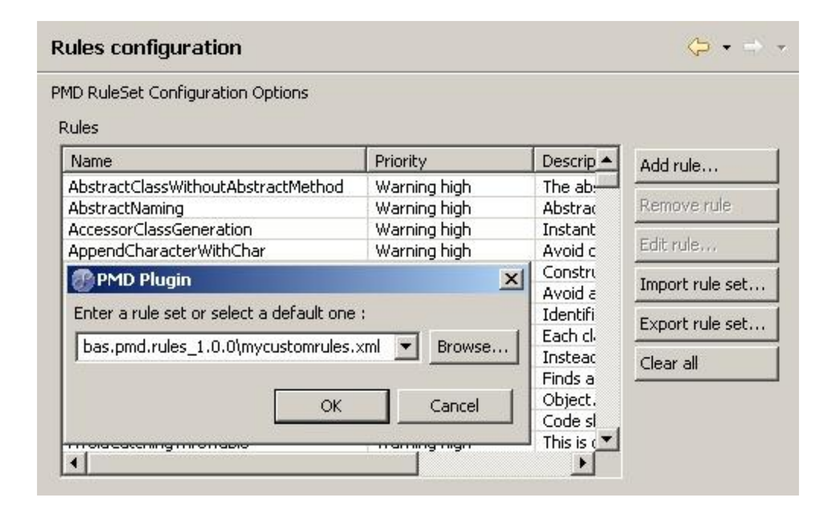



**Instalar el fragmento**

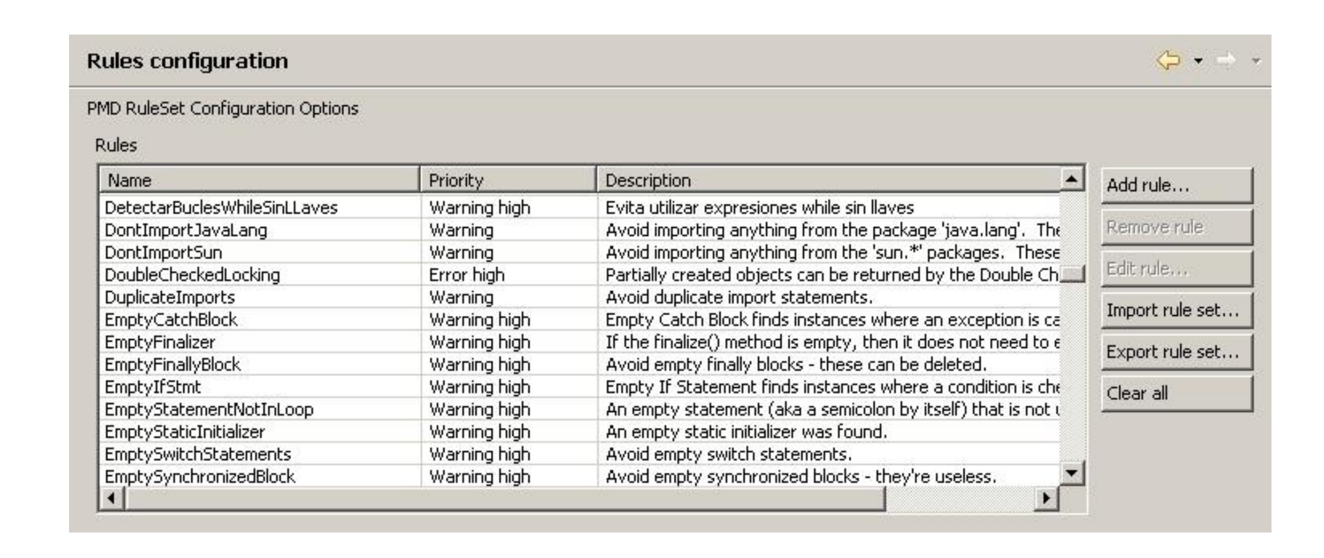
Para instalar el fragmento descomprimiremos la distribución del paquete en el directorio *plug-ins* del Eclipse y lo reiniciaremos.




Sólo queda, entonces, activar el conjunto de reglas personalizadas y para ello, como ya hemos visto, abriremos *Window >> Preferences* y seleccionaremos *Rules configuration* dentro de la categoría *PMD*. Pulsaremos sobre el botón “Import rule set…” y seleccionaremos el archivo XML de definición del conjunto de reglas creado para el ejemplo. Como último paso pulsaremos sobre el botón ”OK”.




Podemos observar cómo aparece la nueva regla creada (*DetectarBucleWhileSinLlaves*) en la lista de reglas. Así, cada vez que PMD realice una validación tendrá en cuenta la nueva regla que hemos introducido.


Rules configuration
PMD RuleSet Configuration Options
Rules
Name | Priority | Description
DetectarBuclesWhileSinLLaves | Warning high | Evita utilizar expresiones while sin llaves
DontImportJavaLang | Warning | Avoid importing anything from the package 'java.lang'. The
DontImportSun | Warning | Avoid importing anything from the 'sun.*' packages. These
DoubleCheckedLocking | Error high | Partially created objects can be returned by the Double Ch
DuplicateImports | Warning | Avoid duplicate import statements.
EmptyCatchBlock | Warning high | Empty Catch Block finds instances where an exception is ca
EmptyFinalizer | Warning high | If the finalize() method is empty, then it does not need to e
EmptyFinallyBlock | Warning high | Avoid empty finally blocks - these can be deleted.
EmptyIfStmt | Warning high | Empty If Statement finds instances where a condition is che
EmptyStatementNotInLoop | Warning high | An empty statement (aka a semicolon by itself) that is not u
EmptyStaticInitializer | Warning high | An empty static initializer was found.
EmptySwitchStatements | Warning high | Avoid empty switch statements.
EmptySynchronizedBlock | Warning high | Avoid empty synchronized blocks - they're useless.
Add rule...
Remove rule
Edit rule...
Import rule set...
Export rule set...
Clear all

# 5 Tareas ant en servidor

La validación del código java con PMD también es posible ejecutarla en el servidor como tareas Ant. Así se asumen los siguientes acuerdos:

- El fichero de configuración de PMD de cada aplicación se denomina “**PMDconfig.xml**” y se encuentra en **“/dominio_wls8/j2se/jakarta-ant-1.5.3-1/rules”**. Va acompañado de la carpeta **“rulesets”**.

- Los ficheros de salida resultado de la validación serán, “**pmd_report.xml**” y “**pmd_report.html**”, en formato xml y html respectivamente, y se ubicarán en “**/aplic/aaa/java/javs/aaaEAR/aaaTestingWar/test/informes**”. (Donde aaa es el código de aplicación).

- La tarea que ejecuta PMD se denomina “**pmd**”:

  ant pmd

# 6 Anexo 1: Ejemplo

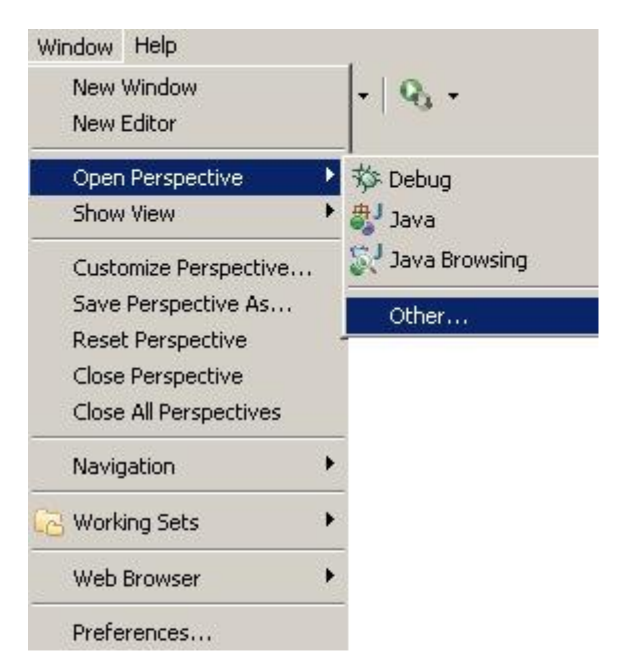
Este ejemplo realiza la verificación del código de *s73aPetStoreWar* a partir del archivo de reglas *PMDconfig.xml* y muestra los resultados obtenidos a través de un informe HTML.

A continuación se realiza la búsqueda de código redundante (duplicado) y se muestra el informe *txt* generado.

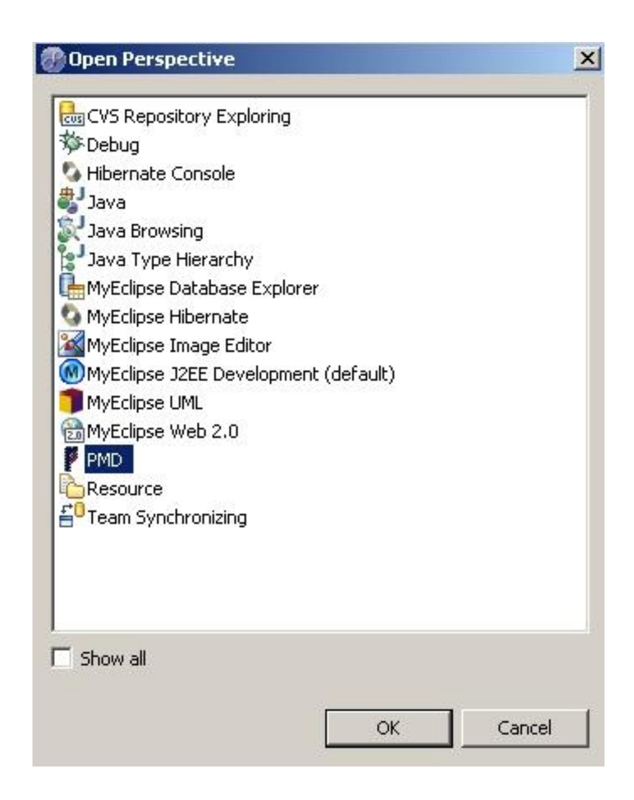
## 6.1 Resolución

Los primeros 3 pasos sólo son necesarios para trabajar con la interfaz más adecuada para la verificación de código por parte de PMD.

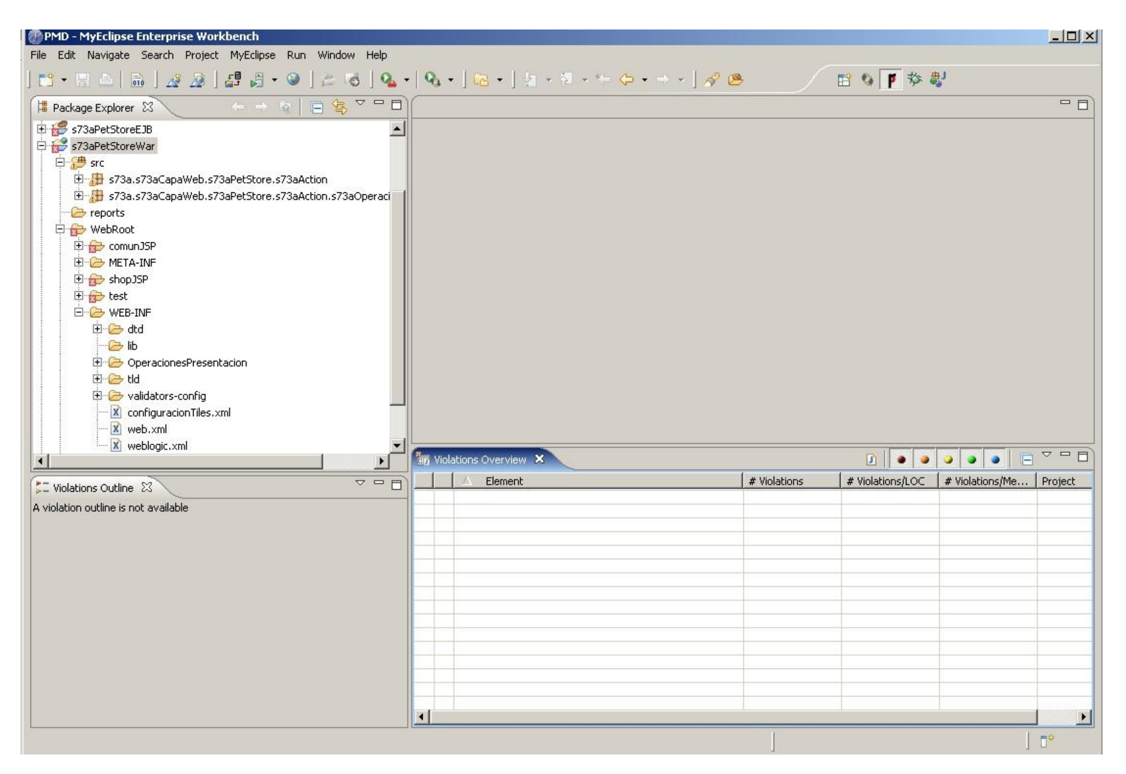
1. En caso de no tener abierta la perspectiva que ofrece PMD habrá que seleccionarla desde la opción de menú *Window > Open Perspective > Other…*




2. En la nueva ventana aparecida se seleccionará la correspondiente perspectiva de PMD como muestra la imagen siguiente.

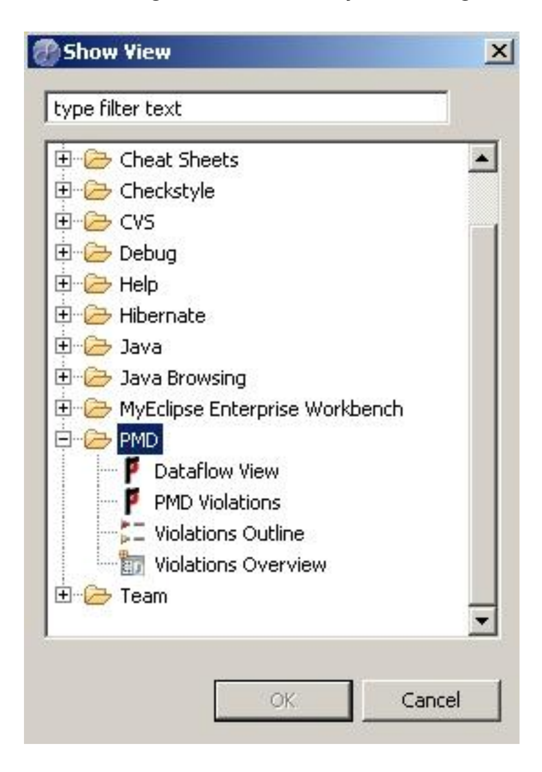



Se trabajará sobre esta perspectiva para poder obtener toda la información relativa a la verificación de código que se desee realizar con PMD.

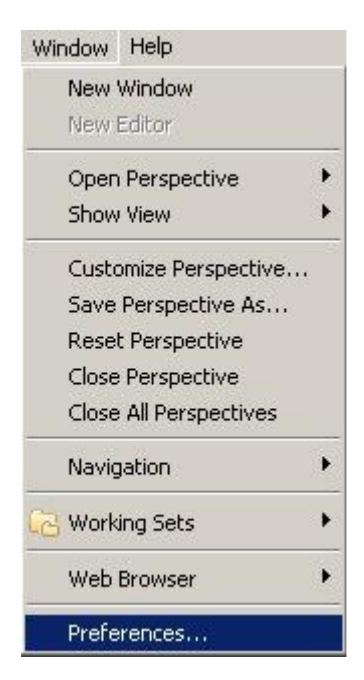


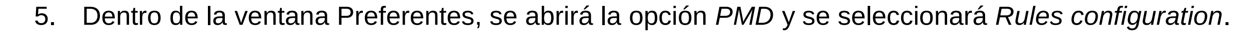

3. Se seleccionarán las vistas que se estimen oportunas pertenecientes a la perspectiva de PMD.

La imagen superior muestra las vistas *Violations Outline* y *Violations Overview*. En caso de necesitar alguna otra se seleccionará la opción de menú *Window > Show View > Other…* Se abrirá la siguiente ventana y se escogerán las vistas deseadas.

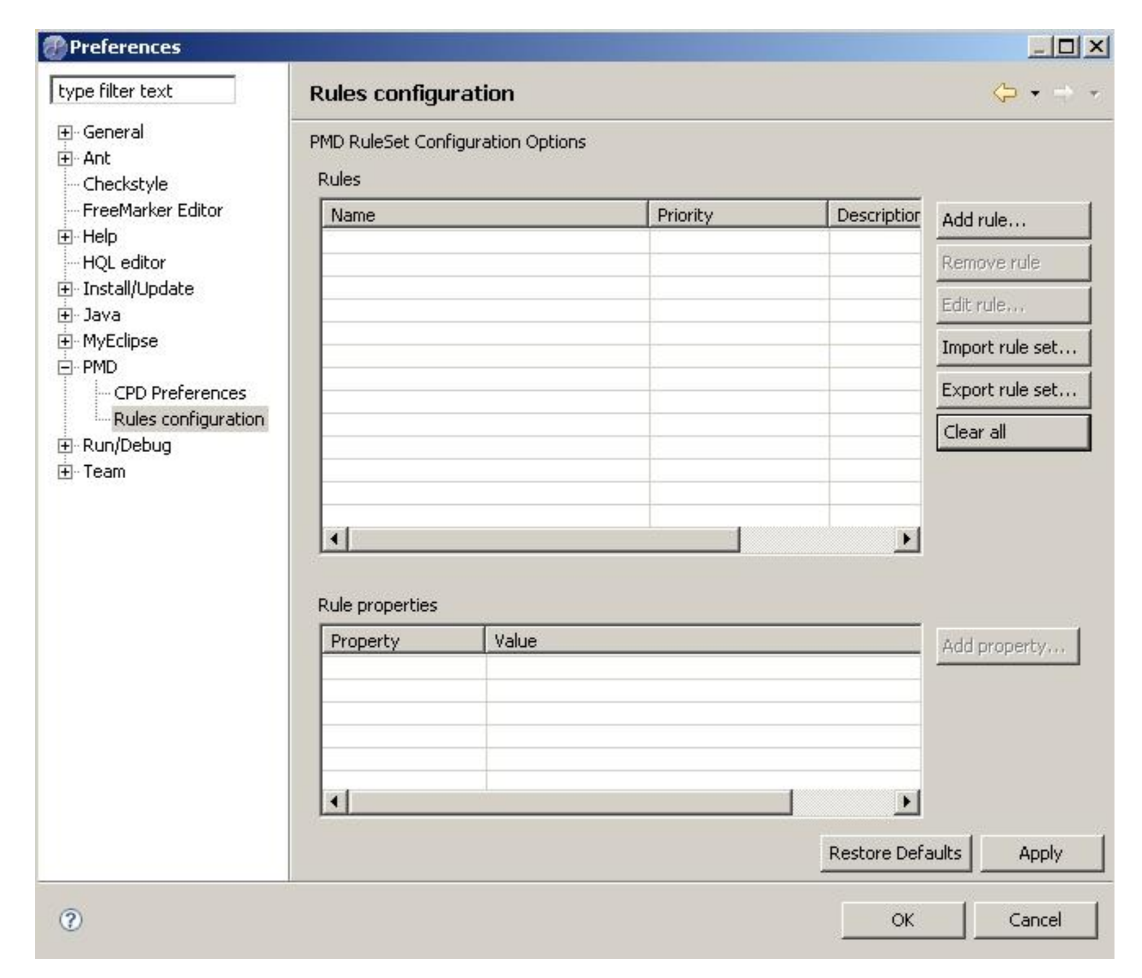



4. Es el turno de configurar las reglas a utilizar con PMD. Para ello se selecciona la opción del menú principal *Window > Preferences…*

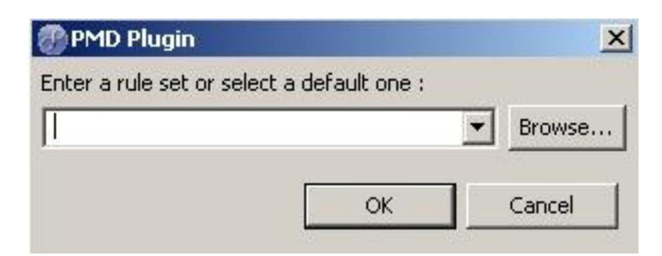




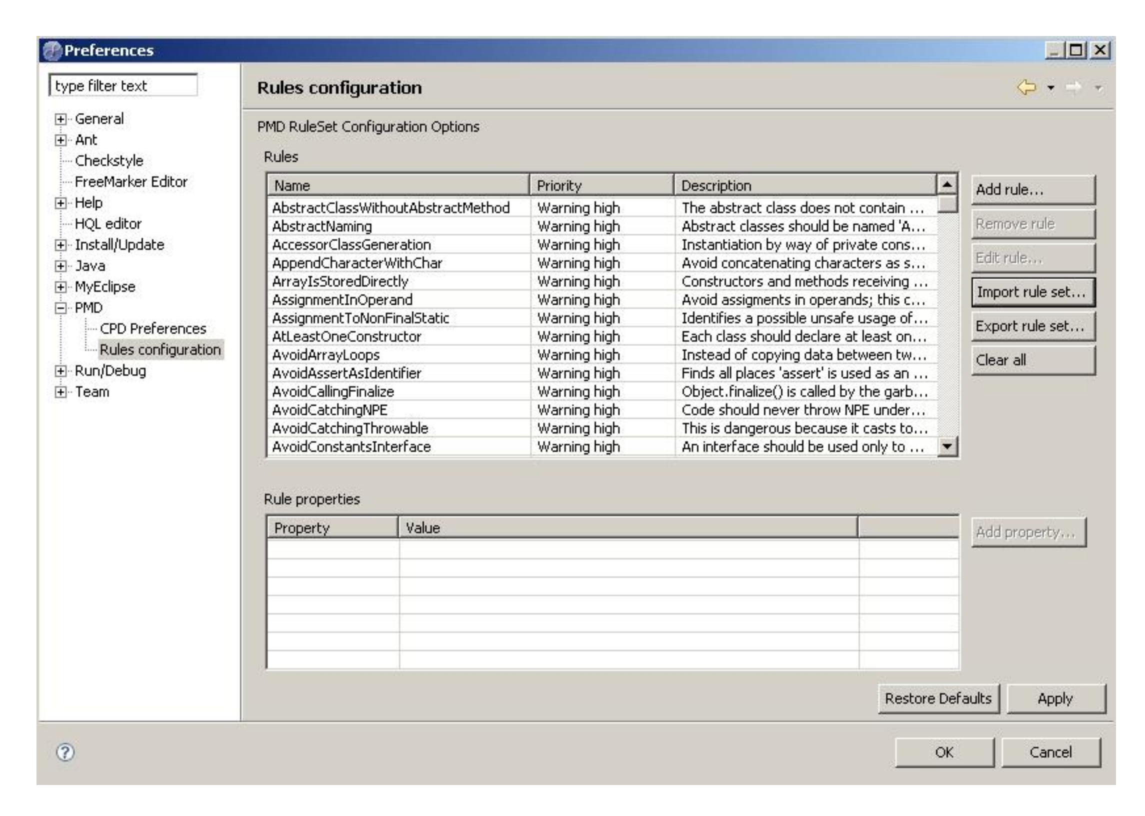
5. Dentro de la ventana Preferentes, se abrirá la opción *PMD* y se seleccionará *Rules configuration*.



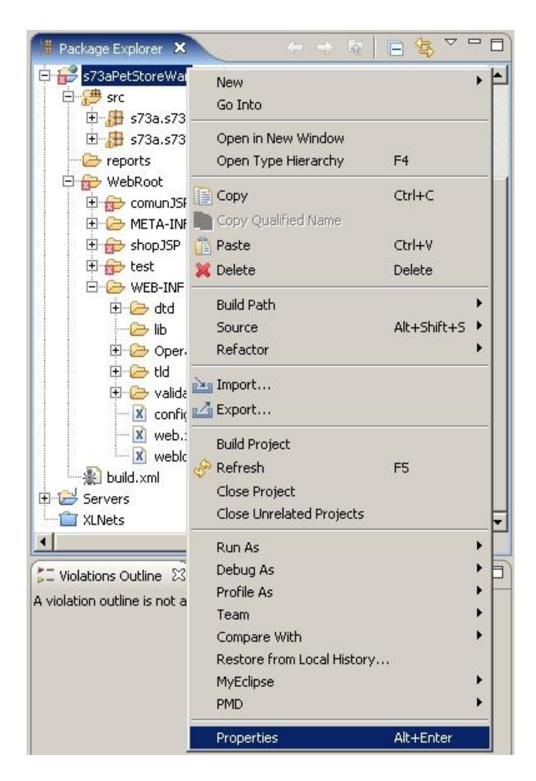

6. Se pulsará el botón *Import rule set…* para poder seleccionar el archivo de reglas deseado. Se abrirá la siguiente ventana en la que se pulsará el botón *Browse…* para buscar el archivo de reglas que se quiere utilizar (*PMDconfig.xml*) o bien se escribirá directamente su ruta en el desplegable y se pulsará el botón *OK*.



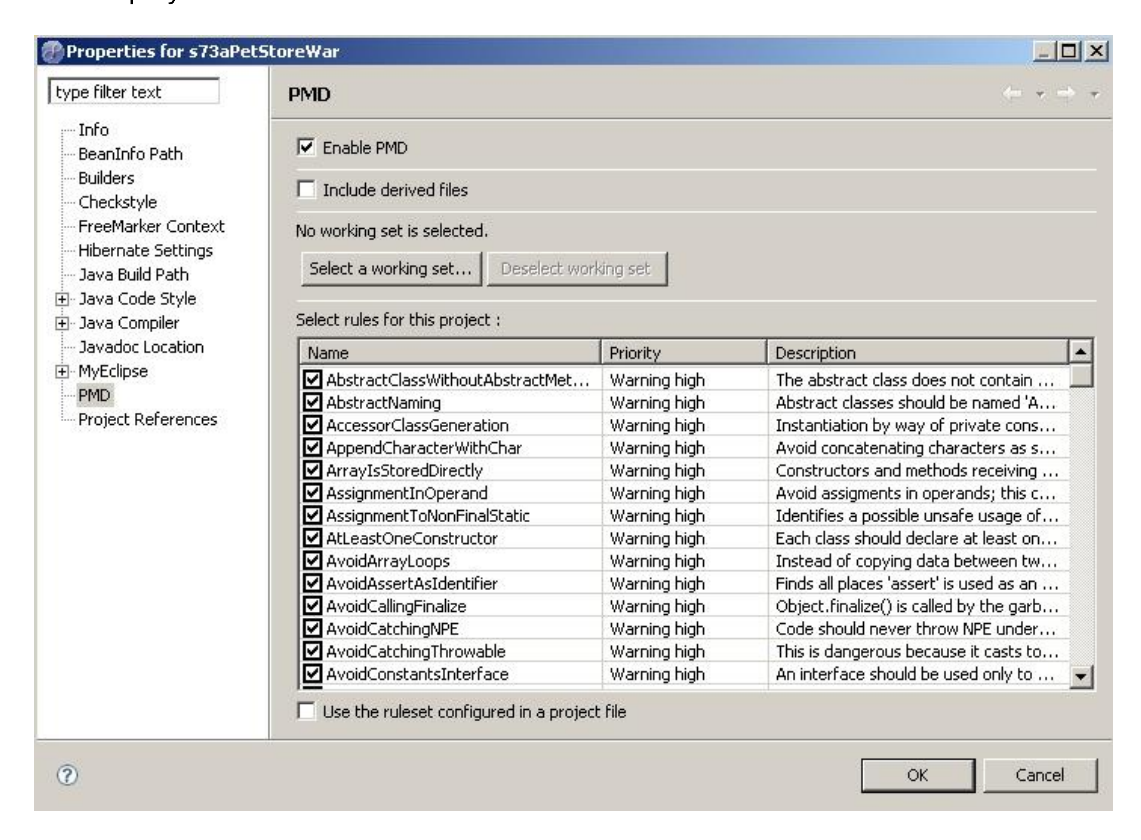

7. Una vez seleccionado el archivo de reglas para el ejemplo PMDconfig.xml el aspecto de ventana de configuración de las reglas de PMD será el siguiente:




8. Ahora se podrá habilitar la ejecución de la verificación del código por parte de PMD por cada proyecto a la hora de realizar la construcción de los mismos.

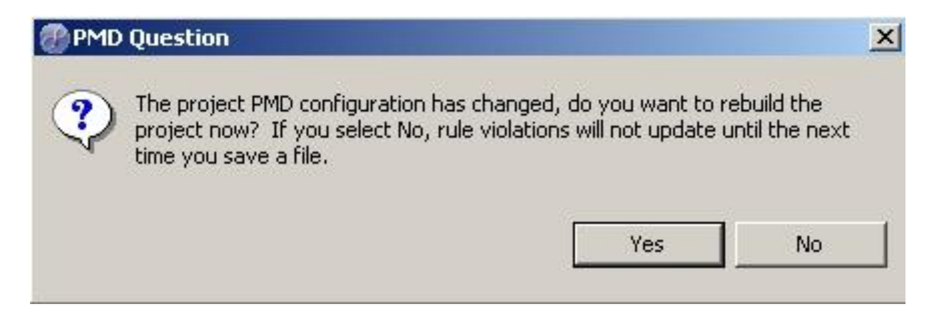


En este ejemplo se habilitará PMD para el proyecto *s73aPetStoreWar*. Entonces, se seleccionará dicho proyecto y a través de su menú contextual (botón derecho del ratón) se seleccionará la opción *Properties* como muestra la imagen superior. Se mostrará la ventana *Properties for s73aPetStoreWar*, que muestra las propiedades del proyecto seleccionado, y se activará la casilla *Enable PMD*. Dejaremos activas todas las casillas correspondientes a cada una de las reglas existentes para realizar la verificación del código considerando todas y cada una de ellas. Podrían considerarse diferentes reglas en función del proyecto a verificar.

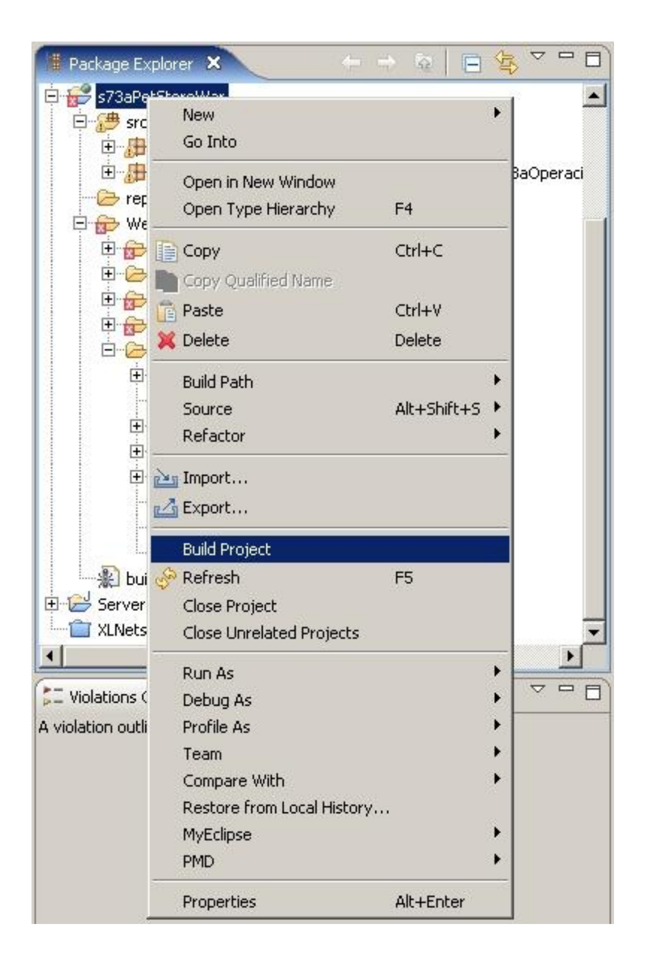



En este momento sólo quedará reconstruir el proyecto para que se lleve a cabo la verificación del código.

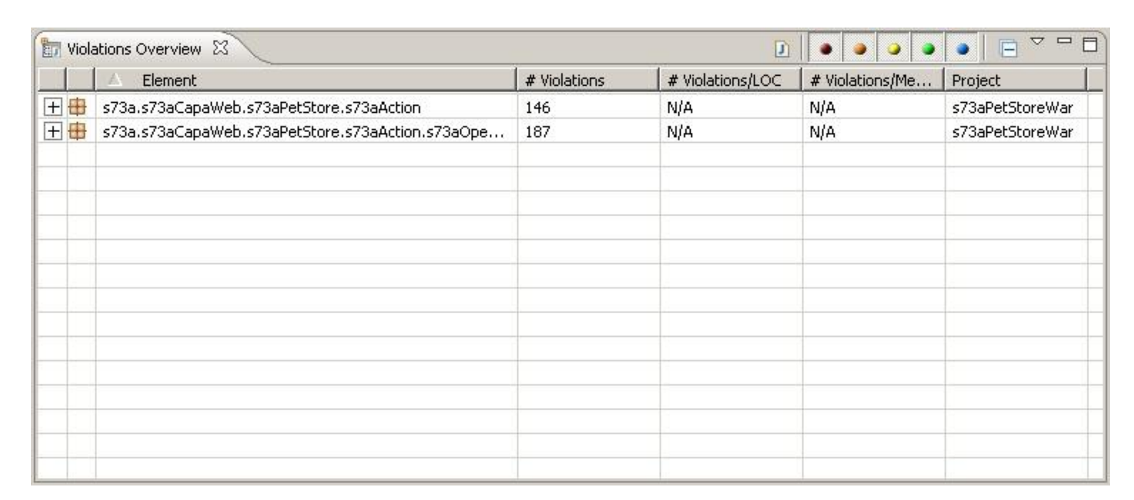
9. Se pulsará el botón *OK*. Entonces, o bien PMD mostrará el siguiente mensaje para encargarse de realizar la reconstrucción automática del proyecto:



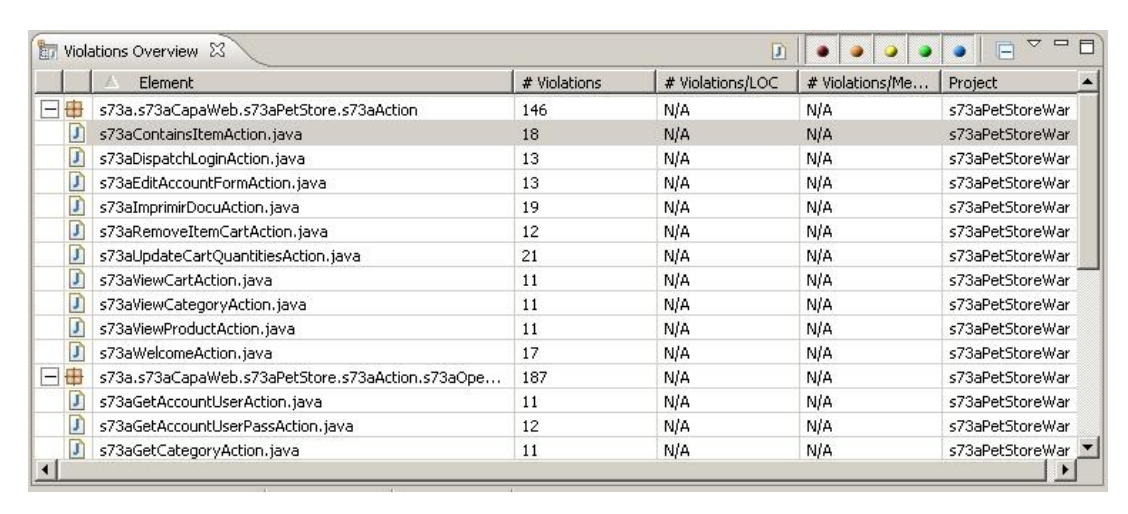

O bien habrá que realizar a mano la reconstrucción. Para ello habrá que seleccionar la opción del menú contextual *Build Project* relativa al proyecto seleccionado:



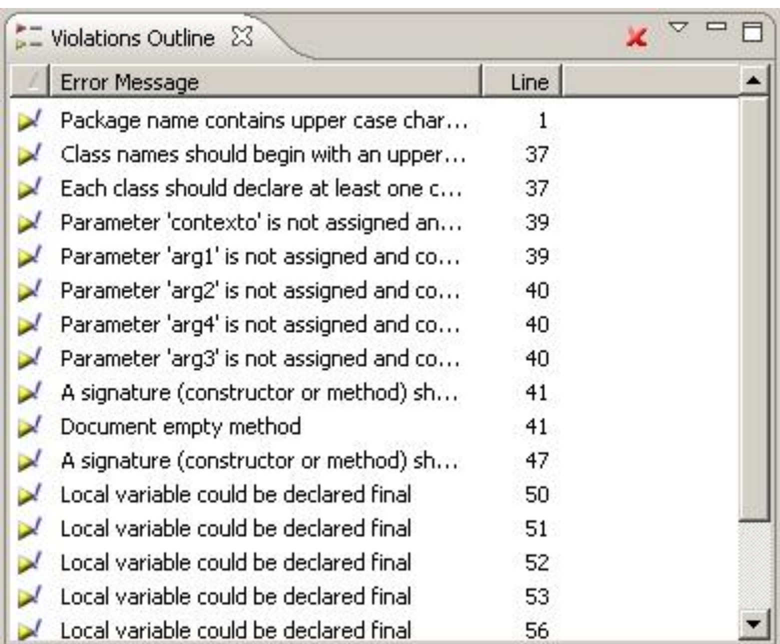

10. Se podrá obtener una vista general de los resultados obtenidos tras la verificación del código en la vista *Violations Overview* como muestra la figura siguiente:




Desplegando en esta vista los elementos en los que se han encontrado violaciones de las reglas consideradas durante la verificación del código y seleccionando clases concretas, se podrá obtener información detallada de las violaciones resultantes con ayuda de la vista *Violations Outline*.

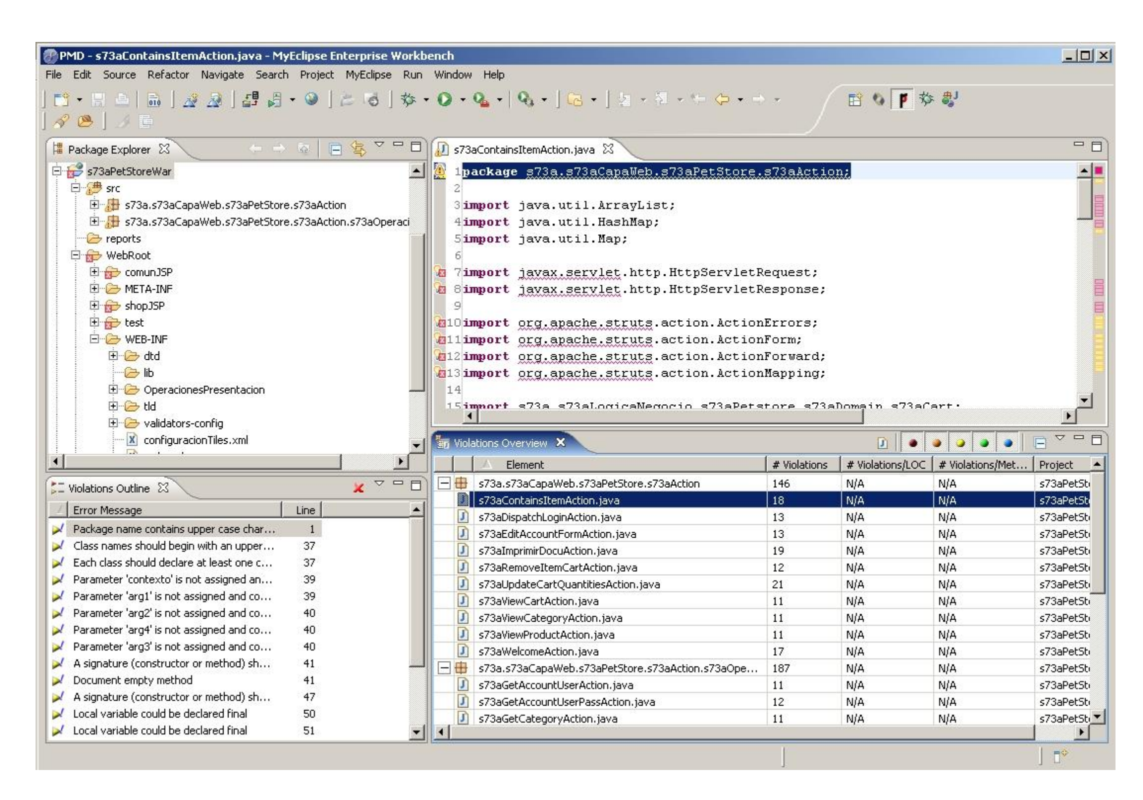

Violations Overview

| Element | # Violations | # Violations/LOC | # Violations/Me… | Project |
|---|---|---|---|---|
| s73a.s73aCapaWeb.s73aPetStore.s73aAction | 146 | N/A | N/A | s73aPetStoreWar |
| s73aContainsItemAction.java | 18 | N/A | N/A | s73aPetStoreWar |
| s73aDispatchLoginAction.java | 13 | N/A | N/A | s73aPetStoreWar |
| s73aEditAccountFormAction.java | 13 | N/A | N/A | s73aPetStoreWar |
| s73aImprimirDocuAction.java | 19 | N/A | N/A | s73aPetStoreWar |
| s73aRemoveItemCartAction.java | 12 | N/A | N/A | s73aPetStoreWar |
| s73aUpdateCartQuantitiesAction.java | 21 | N/A | N/A | s73aPetStoreWar |
| s73aViewCartAction.java | 11 | N/A | N/A | s73aPetStoreWar |
| s73aViewCategoryAction.java | 11 | N/A | N/A | s73aPetStoreWar |
| s73aViewProductAction.java | 11 | N/A | N/A | s73aPetStoreWar |
| s73aWelcomeAction.java | 17 | N/A | N/A | s73aPetStoreWar |
| s73a.s73aCapaWeb.s73aPetStore.s73aAction.s73aOpe… | 187 | N/A | N/A | s73aPetStoreWar |
| s73aGetAccountUserAction.java | 11 | N/A | N/A | s73aPetStoreWar |
| s73aGetAccountUserPassAction.java | 12 | N/A | N/A | s73aPetStoreWar |
| s73aGetCategoryAction.java | 11 | N/A | N/A | s73aPetStoreWar |

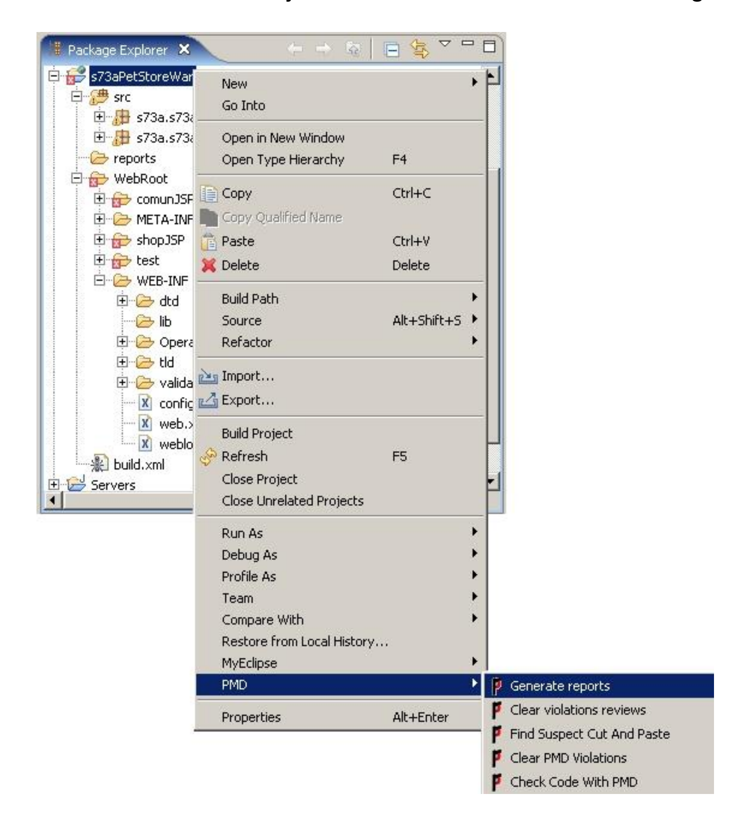

Violations Outline

| Error Message | Line |
|---|---|
| Package name contains upper case char… | 1 |
| Class names should begin with an upper… | 37 |
| Each class should declare at least one c… | 37 |
| Parameter 'contexto' is not assigned an… | 39 |
| Parameter 'arg1' is not assigned and co… | 39 |
| Parameter 'arg2' is not assigned and co… | 40 |
| Parameter 'arg4' is not assigned and co… | 40 |
| Parameter 'arg3' is not assigned and co… | 40 |
| A signature (constructor or method) sh… | 41 |
| Document empty method | 41 |
| A signature (constructor or method) sh… | 47 |
| Local variable could be declared final | 50 |
| Local variable could be declared final | 51 |
| Local variable could be declared final | 52 |
| Local variable could be declared final | 53 |
| Local variable could be declared final | 56 |

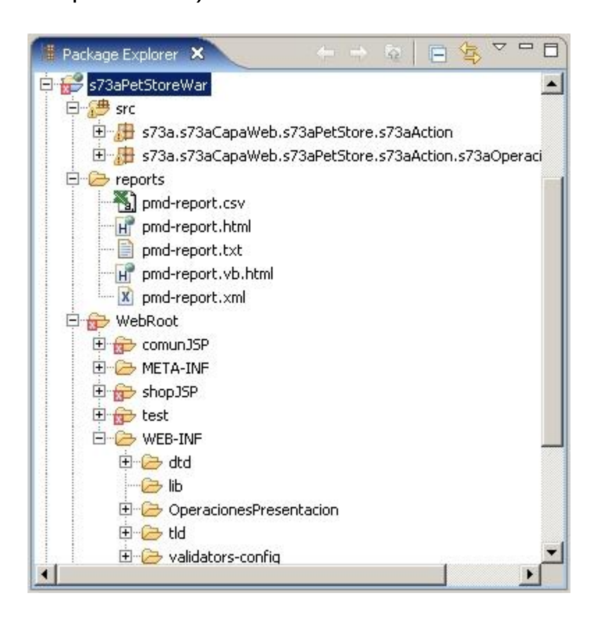
A su vez, seleccionando los mensajes de error de esta vista (vista *Violations Outline* representada en la imagen superior) se accederá al punto exacto de la clase donde se ha cometido dicha violación. La imagen siguiente resalta la instrucción de la clase *s73aContainsItemAction* correspondiente al primer mensaje de error “Package name contains upper case characters”.



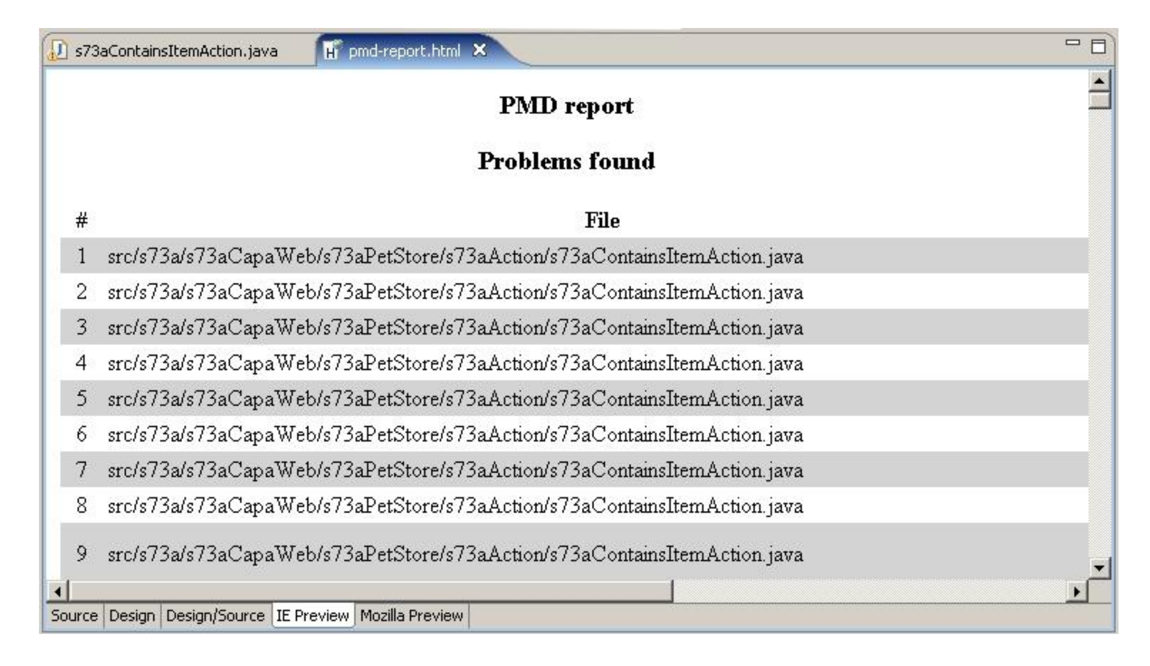
11. Es el turno de generar los informes relativos a las violaciones de las reglas consideradas encontradas tras la ejecución de la verificación del código.



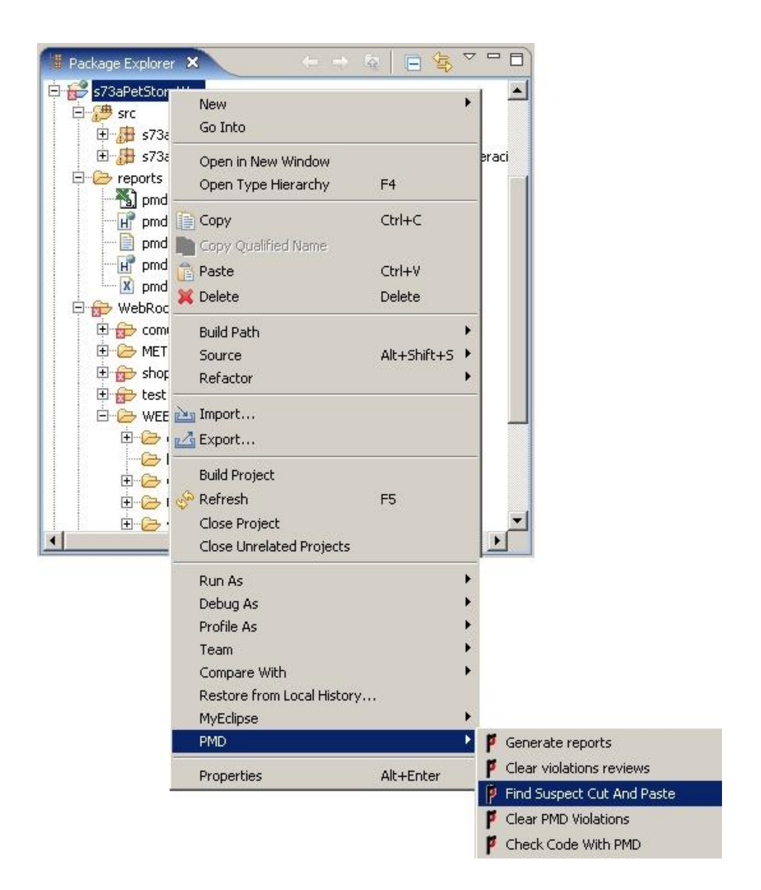
Para ello, de nuevo, se hará uso del menú contextual relativo al proyecto considerado (como muestra la imagen superior) y se seleccionará la opción *PMD > Generate reports*. Se podrá observar que se han creado automáticamente los informes correspondientes dentro de la carpeta *reports* (carpeta destinada para tal fin).



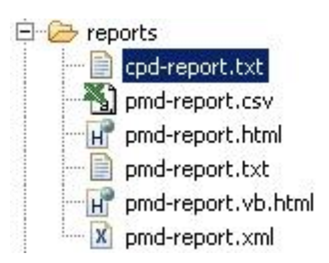

12. Abriendo el informe generado *pmd-report.html* y seleccionando la pestaña *IE Preview* se observará el contenido del mismo:




13. Ahora se va a tratar otra opción que se encuentra disponible al trabajar con el plug-in PMD: la búsqueda de código redundante (duplicado). Para ello se vuelve a desplegar el menú contextual correspondiente al proyecto considerado y se selecciona la opción *PMD > Find Suspect Cut And Paste*.




Se generará un informe en formato de texto plano (*cpd-report.txt*), que también se ubicará dentro de la carpeta *reports*.




14. Abriendo dicho documento se podrá observar que el informe generado muestra el código que aparece duplicado (puede estar duplicado dentro de la misma clase o en varias clases distintas) e indica la clase a la que pertenece dicho código duplicado y a partir de la línea en la que aparece.

s73aContainsItemAction.java | cpd-report.txt

```
=====================================================================
Found a 38 line (140 tokens) duplication in the following files:
Starting at line 45 of C:\PETworkspace\s73aPetStoreWar\src\s73a\s73aCapaWeb\s73aPetSt
Starting at line 46 of C:\PETworkspace\s73aPetStoreWar\src\s73a\s73aCapaWeb\s73aPetSt
        if(account.get("ADDR2")==null || ((String) account.get("ADDR2")).equals(""))
        {
            account.put("ADDR2",s73aAbstractDao.CONSTANTE_NULO_STRING);
        }

        String userName=(String)contexto.get("username");

        account.put("USERID",userName);

        account.put("PASSWORD",userName);

        contexto.set("account", account);


        super.preProcessMappingEntrada(contexto, arg1, arg2);
    }

    /*
```